AD NUMBER

AD800387

NEW LIMITATION CHANGE

TO

Approved for public release, distribution unlimited

FROM

Distribution authorized to U.S. Gov't. agencies and their contractors; Critical Technology; SEP 1966. Other requests shall be referred to Air Force Rome Air Development Center, Attn: EMLI, Griffis AFB, NY 13440.

AUTHORITY

RADC ltr, 9 Jan 1970

STUDY OF ASSOCIATIVE PROCESSING TECHNIQUES

R. M. Bird
J. L. Cass
Richard H. Fuller
et al

AFLC, GAFB, N.Y., 18 Oct 66-154

FOREWORD

This final report was prepared under Contract AF30(602)-3756, Project 5581, Task 558109, during the period of approximately May 1965 to March 1966. The RADC Project Engineer was Morris A. Knapp (EMIID).

The study effort was conducted by the Librascope Group of General Precision, Inc., and specifically was performed by the Advanced Technology Center, Glendale, California. Librascope personnel contributing to the study task or to the report include: R. M. Bird, J. L. Cass, P. E. Tanner and Dr. J. C. Tu. Principal Investigator was Dr. Richard H. Fuller.

Release of subject report to the general public is prohibited by the Strategic Trade Control Program, Mutual Defense Assistance Control List (revised 6 January 1965) published by the Department of State.

This technical report has been reviewed and is approved.

Approved: FRANK J. ROMAINI
Chief, Info Processing Branch

Approved: ROBERT J. QUINN, JR., COLONEL, USAF
Chief, Intel and Info Processing Div.

FOR THE COMMANDER: IRVING L. GABELMAN
Chief, Advanced Studies Group

ABSTRACT

This report is in two volumes and describes results of a study of associative processing techniques performed by Librascope Group of General Precision Inc. under RADC contract AF 30(602)-3756. Volume I is an unclassified document and contains all the material in the report except that concerning the ELINT reconnaissance problem which is contained in Volume II. Volume II is a classified document at the SECRET level.

Two associative processors are formulated and evaluated. The first processor, named the associative file processor (AFP), uses a small associative memory, in conjunction with a large head-per-track disc file, to affect content search and retrieval from large formatted files stored on the disc. The processor also allows efficient updating of large, highly dynamic data bases. The applications of AFP in the manipulation of existing sea surveillance files and command-control data bases are investigated in detail. In these tasks, the processor offers a reduction of several orders of magnitude in query response times relative to both presently used equipment and previously proposed associative techniques. A query language, presently used in a command and control system, is shown to be suitable for use with this processor.

The second processor, named the associative parallel processor (APP), is optimized for simultaneous processing of many data elements as they reside in memory. This processor is evaluated for use in the real-time solution of large network-flow or resource-allocation tasks, and also for the real-time reduction of ELINT reconnaissance data. It is shown that, for large problems of either type, the processor offers order-of-magnitude reductions in solution times over conventional methods. Improvements in solution times are such that real-time solutions of important tasks in each problem area are feasible.

All processor configurations presented are based on mechanizations which can be implemented in a practical and economic manner with presently available electronic circuits and memory components. Other applications of these techniques are recommended and further development and study areas are suggested.

CONTENTS

CONTENTS (continued)

ILLUSTRATIONS

1.0 INTRODUCTION

This report appears in two volumes and describes results of a study of associative processing techniques performed by the Librascope Group of General Precision Inc., under RADC contract AF 30(602)-3756. The study herein reported is a continuation of a previous effort funded under RADC contract AF 30(602)-3371[1,2]. During the previous study, Librascope formulated two associative processors. The first processor was optimized to the task of pattern recognition; the second processor was optimized for retrieval of documents, classified by a set of descriptors such as are used in the Defense Documentation Center (DDC) retrieval system. The present study has further refined each processor organization, has extended the defined range of application for each processor and has provided quantitive performance evaluations for each application investigated.

Recent years have seen intensive investigations of digital computer organizations, command sets, and usage methods which represent distince departures from conventional Von Neuman techniques. Much of this effort has as objectives the formulation and efficient solution of non-numeric problems for which conventional techniques are not effective. Complex non-numeric problems frequently allow partitioning of the computational task into many independent subtasks, which may be allocated over many identical machine elements with interchange of results among machine elements. This, together with rapidly developing capabilities for batch fabrication and interconnection of computer components, has led to investigation of distributed or parallel computing networks consisting of many identical computing cells, interconnected in an iterative structure. Examples of such machines include the Holland machine, the Solomon computer and various pattern recognition devices such as the ILLIAC III computer. The associative processors described in this report are also machines of this class.

We feel that the promise of associative processing techniques lies more in their ability to implement novel and demanding machine requirements, implied by non-numeric computing tasks, than in their ability to speed solution of presently solvable problems. Development of novel cellular organization allows machine solution of important new classes of problems, in addition to solution of larger problems of presently solvable types. Accordingly, we have developed associative techniques for content search and retrieval from large formatted data files, for adaptive pattern recognition, for ELINT pulse train sorting and for solution of network flow problems.

The file processing task is implemented by an associative file processor (AFP) which is a variant of a more restricted machine developed under the previous contract for retrieval of Defense Documentation Center (DDC) documents. The data bases, taken as models for the file processing task, were the present Navy Sea Surveillance Data Base and the present Headquarters ASAF (473-L) Command and Control Data Base. In Chapter 2 of this report, the AFP is structured and programmed for file manipulation tasks required by these problems. A user-oriented query language, suitable to each problem environment is described. The efficiency of AFP in this application is evaluated relative to that of presently used equipment and to that of the Goodyear Associative Processor (GAP), now under development for RADC.

The associative pattern processor, developed by Librascope under previous RADC contract, evaluates threshold logic functions by techniques which may be readily extended to transform all or any part of memory contents according to any boolean function of memory contents and external variables. The organization, logic, and command structure for this processor are thus applicable to a variety of other problems which allow parallel processing. In the sequel, we term this processor the "Associative Parallel Processor", APP to denote its general utility.

Chapter 3 of this report described the application of APP to network flow or resource allocation problems. These problems arise in a variety of logistic or tactical environments, in which each of many resources may be allocated to each of many ends, with some cost or return known for each possible allocation. The APP is programmed for optimum solution of several variants of this problem in near real time. Solution rates are compared to those available from state-of-the-art processors of conventional organization.

Applicability of APP to ELINT pulse-train separation is investigated in Chapter 4, which is classified SECRET and appears in a second separate volume. The task is defined, the APP is programmed for its solution, and solution rates are again compared to those available from conventional machines.

These studies collectively illustrate the utility of associative techniques in two broad areas of non-numeric data processing. The first usage area is content search and retrieval from large data bases stored in inexpensive mass-storage media. The second area is simultaneous processing of many data elements as they reside in memory.

Each application area has a distinct machine organization, AFP and APP respectively, best suited to it. The rationale for each machine organization and a summary of its performance in selected applications are presented in the balance of this report. Each of these machines is structured such that it is practically realizable using available technology. For example, the associative memories for these machines can be built using plated wire memory elements described in Appendix C. Timing calculations for the processors are based on this realization.

1.1 ASSOCIATIVE FILE PROCESSOR

Important features of the associative file processor (AFP) are as follows:

1) The data base is stored on a relatively inexpensive head-per-track disc file, since it is felt that true associative storage of large data bases would be prohibitively expensive, both now, and in the foreseeable future. Further economies are provided by using only a single read-write head on each disc track.

2) A large portion of the file (e.g., 1024 tracks) is content searched in a single disc revolution (e.g., 40 ms). The comparison operators: equality, inequality, greater than, less than or bounded search, are each available as search modes. Rapid search is possible through use of an extremely high data rate (i.e., highly parallel) transfer path from the file to a controlling associative memory. Data is transferred simultaneously from many disc tracks to corresponding words in associative memory. This fast transfer path is necessary if search time is not to be obscured by the time to transfer data to the associative memory.

3) The AFP requires only a small associative memory (e.g., 1024 words of 64 bits) to control search of a much larger data base (e.g., 3×10^8 bits). Each bit of the associative memory is a tag bit used to denote some block of disc storage as matching a search criterion. Elements of the data base are never transferred into the associative memory.

4) The AFP performs all operations requisite to efficient manipulation of a large highly dynamic data base. The present Navy Sea Surveillance Retrieval System and the Headquarters ASAF (473-L) Command and Control System were used as models for the data base and for operations on the data base. The AFP performs the search and retrieval tasks and also the file maintenance tasks requisite to these environments.

5) Records are retrieved from the disc or written onto the disc in "gather read, scatter write" fashion. In a single disc revolution, records may be retrieved from many tracks or written into available space on many tracks. For each of these operations, disc accesses are controlled by contents of the associative memory, set during a previous search.

Through its possession of the foregoing features, AFP offers the important advantages of content search and simple memory management, usually associated with list structured files, without incurring the speed disadvantages of sequentially scanning lists and of linkage modifications required to insert or delete records. AFP incurs some storage inefficiency through the use of a fixed record format. However, it is felt that storage efficiency will generally compare favorable with that of list-structured files, due to the elimination in AFP of explicit attribute (field) names, linkage addresses and reference dictionaries.

The data base structure and machine organization for AFP are described in Chapter 2, together with a query language suitable to its intended use. Response times are determined for sample queries derived from the Sea Surveillance task. Comparison of these times to equivalent times developed for the Goodyear Associative Processor (GAP) show that response times for AFP are several orders of magntiude less than equivalent times for GAP. Query response times for AFP are relatively independent of both query content or complexity and file size, neither of which is true for list-organized files or for GAP.

1.2 ASSOCIATIVE PARALLEL PROCESSOR

The associative parallel processor (APP), developed by Librascope under previous RADC contract, was recognized as having utility beyond the picture processing application for which it was designed. Under the present contract, further studies were undertaken to extend the range of defined applicability for this machine and to determine the extend to which the machine organization would vary with problem

environment. Two further applications for APP were investigated. The first is optimum solution of network flow or resource allocation tasks. The second is processing electromagnetic intelligence (ELINT) data. It was found that APP was applicable to each problem with only slight modification and that significant operational advantages acrue to its use. The results of each study are further described.

1.2.1 The Use of APP in Solution of Network Flow Problems

The applicability of an associative parallel processor (APP) to the solution of network flow problems is investigated in Chapter 3. The network flow problem is couched within the model of the so-called Hitchcock-Koopmans transportation problem. Three variations on this model are considered, and are listed in their order of model complexity.

1) Binary assignment problem
2) General assignment problem
3) Transportation problem

An APP is formulated which can solve any of these three variations of this model. The formulation of the APP includes a description of the organization, a description of the command set and the timing requirement for the commands. A comparison of this APP with a previous APP formulated for pattern recognition is made.

Algorithms for the solution of the three variations considered for network flow problems are presented. From these algorithms, programming flow diagrams and detailed APP machine language programs are written which describe how the APP solves these problems. From the flow diagrams and machine language programs, a timing estimate is made to determine the time the APP requires to solve each of these three variations.

To provide a basis for comparison, the solution times are estimated for a serial processor to solve each of these three variations on the network flow problems. Finally, a timing comparison is made between the APP and the serial processor.

From the study presented in Chapter 3, it is concluded that:

1) The APP has a timing efficiency factor of from one to three orders of magnitude over its serial processor counterpart for the three variations of network flow problems considered. The one order of magnitude factor is applicable for smaller matrices, of around 30 × 30 size, whereas the three orders of magnitude factor is applicable for the larger matrices, of around 1000 × 1000 size.

2) The timing requirement for the three variations increases approximately linearly or in idrect proportion to "n", the order of the cost matrix, for the APP, whereas it increases nearly in proportion to "n^3", or the cube of the order of the cost matrix, for the serial processor.

3) The APP can determine optimum solutions for network flow problems, even for large matrices of the order 1000, in milliseconds, which must be considered as virtual real time in respect to human operator reaction time. In contrast, for a matrix of the order 1000, the serial processor requires from the order of several seconds for the binary assignment problem to the order of several minutes for the transportation problem, which certainly cannot be considered as real time for most applications.

4) The basic structure of the APP is identical with that of an APP previously formulated for pattern recognition. However, it is found that several new features are required for matrix manipulation purposes.

5) The matrix manipulating feature of this APP gives it a utility for solving other types of problems with matrix manipulation requirements, including, for instance, linear programming, dynamic programming, matrix inversion and boundary value problems.

6) Many examples of the network flow problem have a virtual real-time processing requirement. The weapon assignment problem and message switching to optimally distribute messages in a complex command and control system are examples. For these problems, the APP can provide real-time solutions, whereas the serial processor cannot. Any additional reasonable cost factors of the APP over the serial processor can certainly be justified for these types of problems.

1.2.2 The Use of APP for Processing ELINT Reconnaissance Data

In Chapter 4 contained in Volume II with a SECRET classification, the ELINT reconnaissance problem is investigated, to determine the feasibility of employing an APP to process this reconnaissance data in real time. The ELINT problem is described, together with an algorithm for its solution.

An APP is then structured and programmed according to this algorithm. The proposed ELINT processing algorithm consists of three cycles, namely:

Mode I	- Pulse train sorting
Mode II	- Emitter Position determination
Mode III	- Verification of emitters

Flow diagrams, machine language programs and timing calculations are given for each of three modes. The most time consuming processing step is the pulse train sorting cycle of Mode I.

A tradeoff study is made between an ELINT APP and a state-of-the-art sequential processor. Cost estimates for each machine are provided. The sequential processor is programmed to perform the ELINT task. Processing times are determined and compared to those for the ELINT APP.

1.3 CONCLUSIONS AND RECOMMENDATIONS

The studies herein reported demonstrate the utility of associative processing techniques for information retrieval and also for parallel processing of data as it resides in associative memory. The associative file processor (AFP) illustrates that small associative memories, used in a control capacity, can greatly facilitate search and retrieval of formatted records from a large data base, and are also of significant use in memory management tasks. Order of magnitude reductions in query response time are obrained with AFP, relative to use of other known retrieval techniques. The use of associative retrieval techniques significantly affects the manner of organizing a data base into a mass file, since files should be fixed format tabular rather than list structured. AFP does not have appreciable effect on the grammar or syntax of a user-oriented query language. It was found, for example, that the present 473-L query language is quite suitable for use with AFP.

Our studies have shown that the organization and command set for the associative parallel processor (APP) are useful with little change for picture processing, for solution of network flow problems and for processing ELINT reconnaissance data. In particular, the associative microcommand, which establishes search criteria for words requiring a particular transformation and defines the required transformation of these words, has direct utility in all three problem areas. This is not surprising, in view of the generality of the concept of "sequential state transformation" upon which the command is based.

The three problems, investigated for solution by APP, are particularly amenable to parallel processing, since each may be solved by algorithms which execute a single operation simultaneously over many data elements. Problems, allowing this high degree of parallelism, will generally be solvable by APP in times appreciably less than for conventional sequential solution. The time gain depends on the number of data elements which may be independently processed. For all problems studied, the speed advantage of APP was one to three orders of magnitude over conventional processing. Associative memories of 1024 to 2048 words are suitable to each application studied.

Librascope studies, together with other similar studies sponsored by RADC, clearly illustrate the importance of integrating an associative memory into the over-all system, rather than adding it as an afterthought. The efficiency of the associative parallel processor is strongly dependent on the close integration of an instruction memory and various search control devices with the associative array. The efficiency would decrease considerably if the associative array were peripherally controlled by a conventional computer through a conventional data transfer path.

For the associative file processor, the important interface is that between the associative memory and the disc file. Unless a highly parallel transfer path is provided at this interface, data transfer time will vitiate any speed gain due to associative search.

The Librascope studies also clearly show the importance of the multiwrite function to effective associative processing. It has long been recognized that a multiwrite capability is requisite to associative parallel processing. Our studies of APP further illustrate the utility of associative parallel processing and thus of the multiwrite function. More important, we show that multiwrite is a necessary feature for effective associative file processing, since it allows realization of the highly parallel transfer path shown necessary for effective file processing.

Based on our studies of associative techniques and on related studies by others, we feel that additional efforts should be undertaken in the following areas:

1) Utilization of present application studies: It is felt that both AFP and APP could be built, using presently available technology to yield significant improvements in present operating systems. An AFP machine of moderate size has been shown by the present study to have real utility for command and control information retrieval. The APP machine allows an important increase in the ability to perform real-time processing of ELINT reconnaissance data. It can also perform complex weapon assignment or other network flow tasks in real time.

2) Further applications studies: Present digital message switching systems represent outgoing queues as list structures in memory. The AFP organization lends itself to associative manipulation of storage. Also, the APP organization seems suitable to quick-look

scanning of space telemetry data in real time. Such scanning would be of significant help in gaining real-time knowledge of monitored events and also in selecting telemetry types for later, more detailed processing.

3) Further development of technology:

a) In particular, more efficient means for performing the multiwrite operation into associative arrays should be developed. The size, cost and power dissipation of semiconductor word electronics, presently available to perform this function, would preclude the function from realization in systems of sufficient capacity for some important applications. We feel that plated wire realizations of multi write word electronics, compatible with plated wire associative arrays, merit further investigation.

b) More effective match storage and match resolution circuitry should be developed. Plated wire devices merit further study for this function as well.

c) Following feasibility demonstrations of the above devices, an experimental associative processor of either the APP or the AFP variety should be built. Operational experience gained with these processors will aid significantly in further development of associative processing technology and in the demonstration of its utility.

REFERENCES - CHAPTER 1

1) Fuller, R.H., Bird, R.M. and Medick, J.N., "Associative Processor Study - Interim Report", DDC No. AD 608-427, October 1964

2) Fuller, R.H., Bird, R.M. and Worthy, R.M., "Study of Associative Processor Techniques - Final Report", DDC No. AD-621 516, August 1965

2.0 ASSOCIATIVE PROCESSING OF FORMATTED FILES

2.1 INTRODUCTION

In developing applications of parallel file processing, it is helpful to review briefly the evolution of data retrieval techniques as they apply to data base structure. The most primitive technique is the use of address calculation with fixed format records. This is of limited applicability, where the data base is dynamic, i.e., continually changing, because of severe memory management problems. Another severe handicap exists in that data retrieval can be done only on the field for which the address can be calculated.

As information science evolved, great progress has been made through the use of list-structured files and serial or sequential list processing. Through list processing, problems were alleviated in memory management, data access and freedom of record format. List processing, however, is time consuming, as by and large, in order to retrieve data, files must be traversed sequentially following the linkages from one record to the next.

List processing enables the use of a very important aspect of data retrieval; namely, content search, which is fundamental because of the inherent association of a transaction or query with the contents of certain records of a file. When a list-structured file is being sequentially processed, record by record, the retrieval criteria can apply readily to record content.

Associative techniques have been explored previously, in an attempt to speed up list processing. However, these efforts have not met with much success because the processing has remained sequential. Any significant speed improvement requires some form of parallel processing, which implies doing a content search over many records simultaneously. As demonstrated in this section, associative techniques are a great asset in parallel processing, provided no exhaustive data transfers are needed, which eat up more time than searching.

The development of data retrieval techniques may be viewed from another evolutionary standpoint, primarily associated with hardware capability. The simplest content search is for a single criterion applied sequentially

to a single record at a time. The next step in complexity is to apply multiple criteria to a single record at a time, i.e., to do a multiple search. As the records are the same for all searches, the speed advantage is the number of searches that logically can be done simultaneously. This is usually not a great number.

More complex yet, because of hardware implications, is the scheme of applying a single search criterion to a multiplicity of records in parallel, which gains a speed advantage directly proportional to the degree of parallelism. This is the level of the parallel file processor described in this section.

The ultimate in speed is achieved by doing a multiple search over a multiplicity of records, which may be effective where the whole data base is a single file, organized for multiple searching. A machine of the latter type applicable to retrieval of fixed format DDC (formerly ASTIA) records, was developed by Librascope under a previous RADC contract [4].

With the objective of doing parallel file processing, it is necessary to examine the data base organization, particularly the record formats. In section 2.2, format constraints, requisite to parallel file processing, are examined. Section 2.3 presents a machine organization suitable for parallel processing of files stored on a head-per-track disc file. Section 2.4 describes a query language suitable to parallel file processing.

2.2 PARALLEL FILE PROCESSING

2.2.1 Introduction

This section presents a critical examination of the current trend in data processing, namely, the serial or sequential processing of list-structured files. It also presents a technique that promises significant advantages in speed, namely, parallel processing by associative techniques, and compares parallel processing with serial processing, both as to applicability and relative advantages. It is concluded that, within the restriction of properly formatting data, parallel processing offers important advantages, not only in lessened retrieval time, but also in relative ease of use for memory management and data access.

2.2.2 List Structuring of Files

There has developed in the Information Storage and Retrieval (IS and R) community a strong tendency to organize large data bases into list structured files[6] That is, the members of each file are identified by being linked or strung together as a list, rather than say, by physical location. Justification for list structuring of files and the use of serial processing for data retrieval is given in terms of:

1) Advantages in memory management.
2) Ready access to all stored data.
3) Responsive representation of the nature of the data.

The current approach to data retrieval from a list-structured data base is based on content search using serial processing. In real-time, dynamic-data-base/ query systems, the need is to retrieve records by content, because of the inherent association of transactions or queries with certain records in the data base. Content search of the data base is natural and efficient, in that it eliminates record position determination.

Hardware and algorithms, as well as data base structure, are oriented toward fast, efficient, serial searches of large blocks of stored data, or else toward rapid path tracing through stored relationships, expressed by linkages, until the desired items are found. This mode of processing is time consuming. It is not expected that

this problem will be alleviated by improvements in conventional computer hardware, since computer system serial processing speeds are rising slowly compared to the increasing need for fast data retrieval from large data bases.

As it is the purpose of this study to determine the applicability of parallel processing to content retrieval within large data bases, it is expeditious to examine the data base structuring in more detail.

2.2.2.1 Memory Management

The term, memory management, applies to the task, usually catagorized as "bookkeeping", of allocating space in storage as it is required. Space may be required when existing dynamic files in the data base acquire new members, or when new files are generated and must be stored. Deleting records or files makes space available, and the efficient use of storage elements requires that means must be provided for reuse of this space.

Dynamic storage allocation is traditionally accomplished via an "available space" list, in which empty data spaces are strung together to form a list, just as is each file. When a new item appears, it is assigned the first cell on the available space list, and newly emptied cells are linked to the end of the list. Thus, the files in the data base may be of indeterminant size without undue inefficiency in storage space requirements.

The available space technique for memory management is particularly effective when the data base is stored in random access memory, that is, when there is no preferential aspect to the storage location of any given item. However, the latter property is not true of currently available bulk memory devices, namely, disc memories. The result is that file entries in the data base get scattered all over the disc, with no concern for the access time required to go from one item in a file to the next, using the linkage that strings the file together. So far, no technique has been developed for assigning available space on a preferential basis, rather than as "next available", so that the

penalty in access time in tracing through a list structured file is just accepted. This access time tends to swamp out computer processing time and to constitute the limit on reduction of data retrieval time.

2.2.2.2 Data Access

It is apparent that access to any desired stored item in a data base can be virtually guaranteed using list structured files. Each file must have associated with it an identification or classification of the contents of the entries in some sort of an index or directory. Then, when a query is presented, the requested data types are matched against the file directories and the proper files are consulted.

Many files do not involve data sets with intrinsic order relationships. At best, in special cases, such as personnel files, one, or a very few order relationships exist. Thus, although an explicit ordering exists in a list structured file, in the usual case of data retrieval, a file entry point at which to start searching cannot be specified. This implies paying the heavy penalty in access time retrieving item after item, assuming the file occupies random disc locations generated by the forementioned techniques of memory management.

In some applications, this relatively slow access is acceptable, particularly where many separate searches or retrievals are concurrent. A successful technique is to queue disc access requests and sort them into time sequence, reducing apparent access time. A strong multiprogramming capability is required, and there is no increase in speed of response to any given request.

Because of the format problems, and to help better express the intrinsic relationships among the data, other structurings have been explored. Specifically the graphic node-relationship form has been studied[7]. In this form, the concept of "entity" is exploited. Each node corresponds to an individual person or thing, or a classification of individuals. The graphical representation consists of laying out all the nodes and then drawing in pertinent directional relationship lines connecting related nodes. In the reference, a comfortably small number of types of relationship lines seem to suffice to represent a diverse amount of data in a data base.

The graphical node-relationship representation of a data base is mentioned, not because of its potential value for parallel processing, for this kind of graph traversal is eminently a serial process, but because of the explicit treatment of "entity" and "relationship" as concepts in date base structuring. Entities, again, are individuals, such as persons, places, or things, or classifications and characteristics of individuals Examples are: Capt. R. Owens, Griffiss Air Force Base, the Valley Forge (individuals), Commanding Officer, Airfield, Aircraft Carrier (classifications), and Age, Elevation, and Maximum Speed (characteristics). The relationships most readily expressed in a node-relationship graph are qualitative: for example, " is an example of","is a component of," " is located at", "is a typical". These relationships apply among various individuals and classifications.

Quantative relationships do not yield so readily to such a graphical representation, as they express values of characteristics as applied

to individuals or classes. A ternary relationship is introduced in the forementioned reference to handle quantitative relationships, with the cumbersome introduction of every value or physical quantity in the data base as a separate node in the graph. Clearly, the difficulty of traversing such a graph for a content search is compounded by this strategy.

From a study of a graphical node-relationship graph representing a data base, a definite conclusion can be drawn, namely, that structuring files as lists of entries is a format choice and not an intrinsic property of the data. It also helps clarify the inherent nature of a data base in distinguishing entities and relationships which will be of use in formatting files for parallel processing.

The preceding sections have shown that serial processing and list structuring of files both have serious limitations, particularly in the speed of processing. They show that neither concept is inherent in a data base structure, or in the processing requirements per se. Therefore, in considering parallel processing concepts to improve file processing speeds, it is not unreasonable to reconsider data formats and choose file structures that enhance parallel processing.

2.2.3 Parallel Processing of Fixed Format Files

The primary purpose of parallel processing of files is to attain a high processing rate, implying short data search and retrieval times. The objective is to make each entry in a file, and, in fact, each field in each entry, equally transparent to a parallel content search, and to establish a correspondingly efficient retrieval of desired data.

Parallel processing of files involves the parallel-by-track reading of an assigned region of a disc memory containing the file of interest, and a simultaneous field-by-field comparison of each bit stream with the search criteria, flagging for subsequent retrieval those records which qualify.

The searchable fields of the records in a file to be processed in parallel must be in a fixed format. That is, like fields must appear in like places from record to record, so that only one field is searched for the required criterion in each bit stream at any given instant of time. This is because the search criteria, derived from the query, generally specify several fields and required values in these fields. A variable format would require looking for all field identification marks and all field values every character time, which would overload any reasonable device.

While all records within a file must conform to the format chosen for the file, there is no constraint, or even preference, with regard to the search, related to the sequence of fields within the record format.

All fields are equally transparent to the search, for example in the

VIP file in the sea surveillance task (See Section 2.2.3.2), it is no disadvantage to have the ship identity field follow the VIP name field, as the ship name field can be searched just as readily as it can be in the ships' file. An important advantage of having all fields transparent to an associative search is elimination of the need to sort files on each search criterion of interest.

In this discussion, as well as in the succeeding hardware description, it is assumed that a single mechanism performs a single search at a time. While duplicate search equipment could, presumably, double the search rate, other, more subtle changes, such as doubling the disc speed, could achieve the same result. Accordingly, the matter will not be considered further.

2.2.3.1 Blocks and Block Flags

Disc memory tracks have a fixed number of bits and it is natural to divide them into a number of sectors or blocks of uniform length. One direct advantage is the ability to use fixed addressing. However, any given block length is not going to suit all of the files in a data base without wasting some space. It is normally necessary to dissociate block length and record length, which is done by making provision for multi-block records. That is, the formatted, searchable portions of file entries may occupy several adjacent blocks. The block length then becomes a somewhat arbitrary parameter and can be optimized for any particular job.

Significant advantages in memory management and data retrieval are concomitant with the assumption of fixed block length. These come about by the use of flags, assigned on a block-by-block basis, and stored elsewhere than in the data storage blocks themselves.

Memory management, that is, the accounting for and allocation of disc storage space as required, is facilitated by the use of an "obsolete" or "empty" flag associated with each block. As no ordering exists in the obsolete flags, space can be assigned at software option. The specific technique which is very effective is to assign to a new record the first (in time) obsolete slot (of one or more blocks) that can be written into within a specified region (i.e., a soft-ware-defined set of tracks) of disc memory. This has three advantages over the use of an available list, namely, access time is reduced, records can be segregated into regions under software control, and the assignment and writing of the record are

purely hardware functions. In the case of multi-block records, an additional header block flag is needed to identify permissive blocks in which to start writing (see section 2.3.6). The obsolete and header block flags can conveniently be stored in dedicated tracks in the disc memory, provided only that they are timed to appear with their associated blocks.

Data access by content search is facilitated by the use of compare flags on a block-by-block basis. Since the parallel search is mechanized as a block-by-block comparison, but records, rather than blocks, qualify and are read, it is necessary to do operations on the compare flags. Additionally, statistics may be wanted after a search, rather than reading the qualifying records. For these reasons, it is of advantage to hold the compare flags in a special memory, external to the disc, in which they can be operated on in parallel.

Thus, the fixed format structuring of a data base for parallel processing, rather than list structuring for serial processing, offers significant advantages for memory management and data retrieval, when combined with suitable hardware elements. Greater control, with less effort, may be exercised by the software in storage space assignment. More significantly, after a parallel search, to which a whole file is transparent, the desired records are directly accessible for reading and further processing.

2.2.3.2 Data Structure and Required File Formats

This section is addressed to some of the problems and difficulties encountered in attempting to structure or lay out a data base in fixed-format, parallel-content-searchable files.

Record versus block length has been treated in the previous section and will not be treated further, except to remark that sometimes field length and processing convenience can be traded off against each other.

Variable-length records arise when file entries contain raw, unformatted data, such as "comments", or when there are multiple tabular entries. The raw data portion of a record is more or less intrinsically unsearchable, and no capability is lost by storing it separately, with a linkage in the formatted part of the record.

When file entries have more than one value for specific position, the format problem has been solved by allowing variable-length, variable-format records. This is done at significant storage expense, for each descriptor value must have the descriptor name associated with it. This technique may be best illustrated by an example taken from the sea surveillance task[5]. In one approach, a ships dynamic file is maintained, which lists among other things, VIP's aboard each ship. As the number of VIP's is variable from ship to ship, the number of VIP fields is variable, and, furthermore, changes from time to time. Since the VIP fields within the given ship's record have non-predictable positions, each must be identified within the record as a VIP field.

The above variable length record, with descriptor name-descriptor value pairs, can be eliminated by recognizing a VIP as an entity. Since there are many VIP's and they all sometimes have the property of being aboard

one ship or another, this constitutes a topic, and a file can be generated. This file will have the VIP as the primary entity, and, as he can, at most be aboard one specific ship, have boarded at one specific time, have one specific function and destination, the file can have a fixed record format.

The preceding example suggests that using the concepts, described in section 2.2.2.3, of entity and relationship, a file in a data base can be looked upon as a matrix. The rows and columns correspond to entities, and the matrix elements themselves to qualitative or quantitative relationships. A measure of the tractability of the data base and of the quality of file design is the number of null elements in the files. This, as does record length, affects primarily the storage efficiency, with little direct effect on data retrieval capability or speed of parallel processing.

An important guide to file structure is to include all possible relationships of the file entries within one file. This avoids ever having to determine the intersection of two searches, which cannot be done in parallel, and is, therefore, a slow, serial process. For example, take the sea surveillance task. It is highly undesirable to require searches on both a ships' dynamic file and a ships' static file in response to a query. These searches will, in general, come up with with two tables of ships' names (as in the query considered in the next section), and the names common to both tables must be found for the query response.

The alternative technique is to maintain a single ships' file with both dynamic and static data in each ship's entry. For parallel processing, some positional relationship of data must exist and it not readily obtained any other way.

The concept of subfiles is also useful in formatting a data base. Drawing on the provious example, a ship's static data may differ widely in content, depending on whether the ship is a passenger, merchant, or naval vessel. However, the dynamic data format may be identical for all ships. Processing techniques can be devised that will apply a search to a subset of file entries or to the whole file as desired.

The preceding discussion strongly suggests that the requirement to organize the files of a data base into rigid fixed format or tabular form may not be as restrictive as it may ar first appear. What restrictiveness there is may well be more than offset by the gain in processing speed by virtue of parallel processing. A by-product advantage of retrieving from fixed format files is that the data is in a natural and effective format for output to an inquirer, or for further processing into a graphic visual display.

2.2.3.3 Processing Rates

To illustrate the effectiveness of parallel processing, as compared with some serial processing techniques proposed, including the employment of associative memories[1], the following approximations to search and retrieval times are derived. It is assumed that the data base resides in disc memory and one or more searches and retrieval occur in response to a query.

Search and retrieval times:

1) Disc memory and computer:

$$T1 = \frac{N}{m} \cdot \frac{T}{2} + N x_1$$

2) Disc memory, computer and auxiliary associative memory:

$$T_2 = \frac{N}{m} \cdot \frac{T}{2} + \frac{N}{m} y + N x_2$$

3) Disc memory with parallel processor and computer:

$$T_3 = (k_1 + 1) T + k_2 N x_3$$

Where:

N is the number of records in the file, or the number of directory entries that must be searched.

m is the number of records per block transferred at a time from disc (in case 2, equal to associative memory size).

T is disc revolution time

(T/2 is the average access time to disc memory).

x_1 is computer serial comparison and processing time per record.

x_2 is computer processing and transfer time per record using the auxiliary associative memory.

x_3 is computer processing time per record.

y is the auxiliary associative memory processing time per block of records.

k_1 is the number of blocks of each record that must be searched.

k_2 is the fraction of the records that satisfy the search.

Comments:

a) If N is very large, it may be appropriate to use less than $T/_2$, such as T/5 or T/10, as disc read requests may be batched and time sequenced; neverthless, with N large, $\frac{N}{m}\frac{T}{10} \gg T$.

b) It has been found that, with small associative memories, $Nx_2 \approx \frac{N}{m} y + N x_2$, and, in either case, the processing time is swamped out by the disc access time so that $T_1 \approx T_2$.

c) T_3 has time $k_1 T$ for search and T for read. If k_2 is reasonably small and, assuming x_1 and x_2 are roughly equal, $T_3 \sim (k_1 + 1)\, T$, independent of N up to relatively large N.

As an example, Averbach[1] considers the sea surveillance data base[5], and the query, "Nearest (in time) ship/aircraft with aircraft aboard with doctor to point x,y". With the assumption of only 32 records in the data base, the disc retrieval time is quoted as being 7.2 seconds, over 35 times the processing time required. It is to be noted that the processing is done largely in an auxiliary associative memory, showing a slight improvement over doing it in the GPC alone. For each additional 32 records in the file, an additional 1.2 seconds retrieval time is needed.

Using parallel processing, by contrast, about two searches and one retrieval process are needed; one search on the ships' static data, to establish doctor and aircraft aboard, and one on ships dynamic data in the same file to determine the ships that are in a region aroung point x,y. This requires three disc revolutions (3T) independent of file size up to several thousand (instead of 32) records per file.

In addition to the search and retrieval time, there is processing time required to find the minimum time in response to the query. This processing time will be small if $k_2 N$ is small; that is, if the search on the ships' dynamic data (for location) is sufficiently delimiting.

For representative disc memories, a revolution time (T) is 50 to 70 milliseconds, establishing the parallel processing time as 0.15 to, say, 0.25 seconds. In the referenced example, the retrieval time is

$$\frac{N}{m} \text{ (access time)} = 1.2 \left(5 + \frac{N}{32} \right) \text{ seconds.}$$

These are compared in the following table:

Records per file N	Serial Retrieval Time	Parallel Processing Time
32	7.2 sec	0.25 sec
1,000	44.4	0.25
10,000	390.	0.5 *

* Assuming the search is $\leq N$ but $\geq N/_2$ records in parallel.

The above examples emphatically illustrate:

a) That serial processing, when disc retrieval times are taken into account, is very slow for real-time search and retrieval, even with moderate sized files and a multi-directory approach; and

b) that parallel processing has relatively high speed and is independent of file size.

2.2.3.4 General Purpose Computer Environment

It is assumed that the parallel processing takes place in a general purpose computer (GPC) environment. That is, the GPC runs the show and makes maximum use of the parallel processing to ease its internal serial processing load. With this, comes the concept of filtering, wherein the search may be too complex for the parallel processing hardware, but the number of candidates for serial processing is grossly reduced. For example, in a sea surveillance system, a search for all ships within one hour travel time of point p involves a great circle distance (GCD) determination and a division by ship speed. At disc rates this is too much of a calculation to expect to do simultaneously on all ship location records, so the GPC would set up a search on latitude and longitude limits, doing the final elimination in core memory.

Other functions left to the GPC include input conversion, generation of search criteria, file purging and updating, generating historical tapes, fallback procedures and output conversion. The requirements on the GPC, then, include a strong multiprogramming capability if all these functions are to be done in real time.

2.2.4 Evaluation of AFP-1

This section is a discussion of the AFP-1 file processor as described in the previous Librascope "Study in Associative Processor Techniques"[1], While AFP-1 offers strong potential in document retrieval applications such as the DDC (formerly ASTIA)Retrieval System , it requires modification to achieve the generality required of command and control query systems. The AFP-1 configuration constituted an intermediate step in this study and served as a vehicle for better understanding. A generalized version of this processor, termed AFP-2, is described in Section 2.3.

An area of cost improvement, achieved in AFP-2, relates to the number of heads per disc track. The AFP-1 system has three heads per disc track. Current disc technology, with the high bit densities available, makes multi-head tracks prohibitively expensive for bulk data storage. In addition to the read head, the AFP-1 has a head slightly downstream to write data in a slot just identified as empty by reading an empty flag. It has another head a farther distance downstream to drop compare flags in records after the records are searched. The AFP-2 design eliminates both of these heads. Records are written with the same head with which they are read.

Since AFP-1 was intended for search over a single homogeneous file, it was structured to allow search over all tracks in a single disc revolution. In the command and control environment, there are usually many files, no more than one of which is searched at a time. It is therefore unnecessary to search all disc tracks simultaneously. In AFP-2, searches are limited to some region of disc memory where the size of a region is determined by storage requirements for a single file.

There are two areas in which the definition of AFP-1 was incomplete. The first relates to the write head necessary to flag empty records. The separation of this head from the read head determines a maximum record length. Many small records could occupy this interval. However, no provision is made to queue the compare flags for these records. A significant increase in AFP-1 complexity would be required to handle these queues. It now seems impractical to store more than one record in the maximum-length slot.

Search criteria in the DDC problem are fixed-length document descriptors. In the general command and control environment, search criteria are normally fields of characters derived from an input query. Conceivably, an AFP could be built in which fields could be recognized, but every possible parameter of every search field would have to be interrogated each character time, thereby throwing an excessive processing load on the associative memory.

In the AFP-1, neither the read nor the retrieval processes were precisely defined. In the AFP-2 system presented in Section 2.3, read and retrieval processes are defined and the machine is structured to allow effective execution of these processes.

2.3 ASSOCIATIVE FILE PROCESSOR

2.3.1 Introduction

This section presents the design of an associative file processor, designated the AFP-2 which accomplishes the parallel file processing tasks discussed in section 2.2. The design is an embodiment of the concept of parallel processing for data retrieval by content using current state-of-the-art hardware.

The AFP-2 description is presented in the context of a command and control application and the design parameters are derived from an existing, successfully operating system* which is limited to serial content search. As such, the AFP-2 design is responsive to all of the requirements of an information storage and retrieval application, involving dynamically changing files in the data base and a user oriented query system for data retrieval. The design is broadly, rather than specifically, application oriented. Consequently, no unique structure is defined; rather, options and tradeoffs are pointed out and discussed.

This section contains a description of the major hardware elements and the individual operations of the AFP-2. Following this, the content search and data retrieval process is discussed. Finally, a number of tradeoffs and options are noted.

The AFP-2 design description in this section demonstrates the feasibility of building within the state-of-the-art a parallel file processor using associative techniques. It shows that the inherent high processing rate advantages can be attained by making use of the parallel processing capabilities of an associative memory while still retaining the relatively low storage cost of a disc memory.

2.3.2 Design Concept of AFP-2

The AFP-2 is designed to do a parallel search on data emanating from a head-per-track disc memory. A large subset of the tracks is read in parallel and the bit streams are compared simultaneously with the search criteria held in separate registers.

---

*Headquarters, United States Air Force Command and Control System (473L).

An associative memory is used to perform the parallel comparison and to store the results as compare flags, assigned on a block-by-block basis. The compare flags, after possible modification by subsequent parallel operations in the associative memory, are used to control, and make efficient, a subsequent data read operation.

The AFP-2 implements a number of data handling operations to allow its performance as a data storage and retrieval system. These, including a multiplicity of search types, are described in section 2.3.4.

Certain design parameters are assumed for illustration, although these are easily subject to modification for a given application. The assumed design parameters are:

1) Blocks of 128 characters of eight bits each.
2) One "obsolete" and one "header block" flag per block.
3) 64 blocks (65, 536 bits) per disc track.
4) 1024 disc tracks (called a section) searched in parallel.
5) As a result of 3) and 4), an associative memory size of 1024 words of 64 bits, plus a few tag bits, each.
6) A total disc memory capacity of several (say 5 or 6) sections (that many groups of 1024 tracks).

Design parameters 1 and 2 are primarily application derived. The 128 character block is used in the operating command and control system noted in section 2.3.1.

Parameters 3 and 6 reflect currently available head-per-track disc technology. Parameter 4 is largely arbitrary, although there is a relation between the block size and the number of tracks searched in parallel due to the structure of the flag tracks.

2.3.3 Summary of Hardware Elements

A summary of the major hardware elements of the AFP-2 follows (Figure 2-1). Each element will be discussed in turn together with its pertinent features.

2.3.3.1 Disc Memory

The primary data storage element is a large head-per-track disc memory. It is organized into a number of sections of 1024 tracks each. (The numbers chosen are primarily illustrative, with some constraints and tradeoffs discussed later.) Each track is divided into 64 sectors, or blocks, of 1024 bits

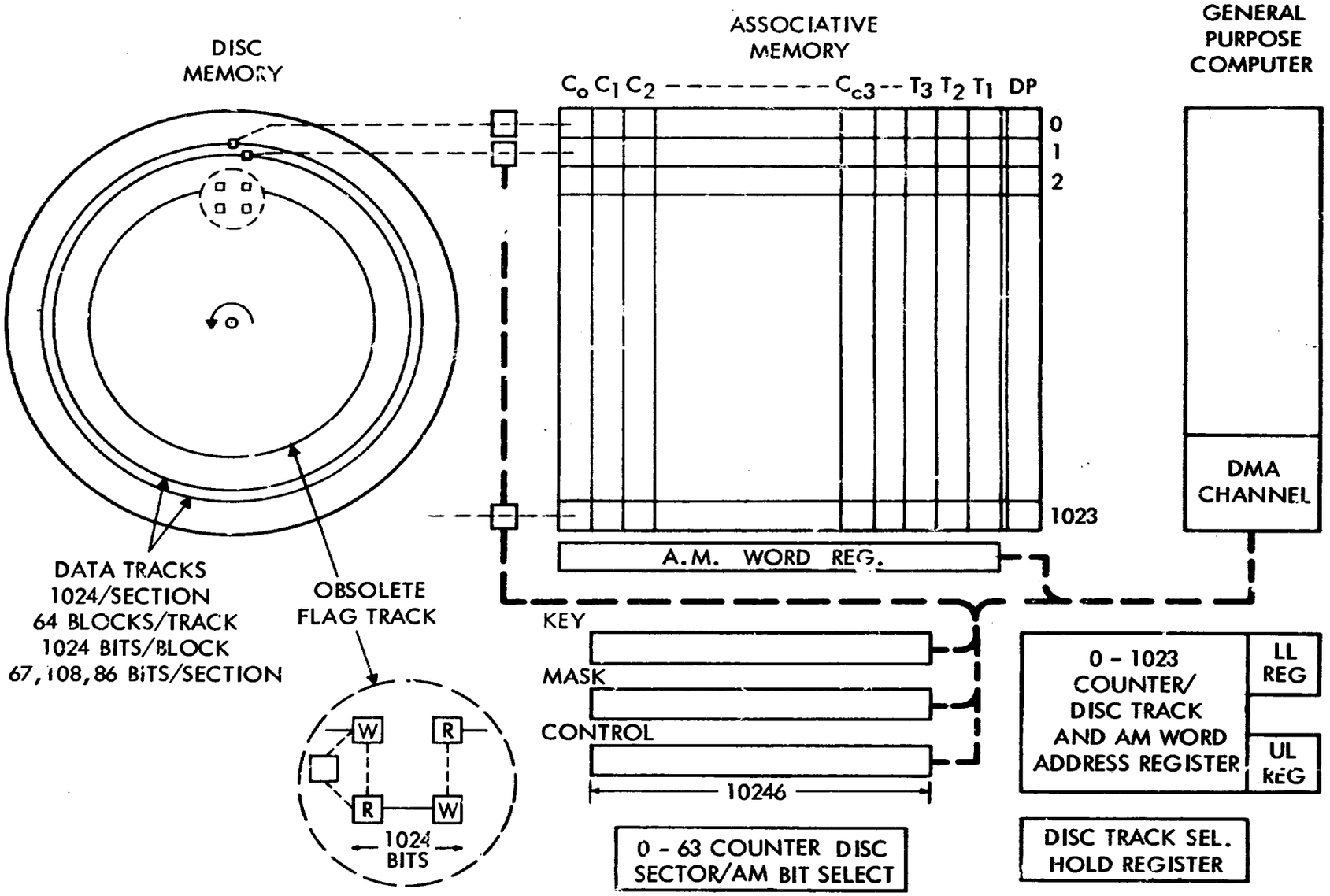


Figure 2-1 AFP-2 Major Hardware Elements

each, which are intended to hold 128 data characters of 8 bits each. This gives a total of 65,536 bits per track. All 1024 tracks of a section are read simultaneously, providing as many bit streams, and are searched simultaneously. Each head has write capability although it is not contemplated to write in more than one track at a time.

Each disc memory section has two additional specialized tracks carrying an "obsolete" (or "empty") flag and a "header block" flag for each block in the section. These tracks also store 65,536 bits each.

The disc memory has the usual accessory features of clock and timing tracks, but for this discussion these will be ignored. Also, there may be inter-block gaps, so the bit counts are to be considered usable bit counts.

2.3.3.2 Associative Memory

The associative memory has the function of establishing and storing "compare" flags for all of the blocks in a disc memory section. It has a word-per-disc memory track (1024), and the word length is the numer of blocks per track (64) plus a few tag bits (Figure 2-1). The exact number of tag bits is determined by some tradeoffs discussed below, but is at least two. A multi-write or write-on-match* capability is assumed.

Consideration will be given to other associative memory features and to the situations in which they are particularly effective. These features include the priority circuit or resolve operation* which selects the "next" qualifying word, and the match type or approximate number of matches operation. There is a difference in the latter two operations, although both give detector plane statistics. The match type operation is a none-one-many distinction, while the other gives an approximate count of the number of matches.

The associative memory has a word register for either-way word transfers, but no mask register as record data never enters the associative memory. It has a bit selecting index register which is also associated with disc block identification.

---

*Terminology used is taken from Reference 3

2.3.3.3 Search Criteria Registers

Three circulating registers, each one block long (1024 bits), are provided to hold the search criteria. These registers operate bit serially in synchronism with blocks on the disc memory, and are designated the key, mask, and control registers. Essentially, their content, mixed with data coming from the disc, including the obsolete flags, serves to fill the associative memory with compare flags.

2.3.3.4 General Purpose Computer

The AFP-2 is assumed to operate in a general purpose computer environment. The AFP-2 is "operated" by the GPC. For efficient data transfer, a direct memory access (DMA) channel is assumed. Other properties of the GPC, such as word length, interact with the AFP-2, but have only a secondary effect on its functioning.

2.3.4 Operations in the AFP

The macro operations performed by AFP-2 may be broadly categorized by two characteristics; namely, the identification of record locations, and what is done to those locations, or the access means and the function, respectively. The functions will be discussed first.

2.3.4.1 Functions

There are five functions related to operations on a data base stored on a disc memory. These are:

1. Write
2. Read
3. Change obsolete status
4. Count
5. Search and set compare flags

2.3.4.1.4 Write - The write function may occur in order to change or add a bit, character, field, or several fields in a record; to change or add a record, portion of a file or even a whole file. Since it is not practical to provide addressing means for individual bits or characters, the write function is offered on a single block, or multi-contiguous block basis. The management problems of trying to string together scattered block locations for writing a multi-block record disfavor hardware implementation over being handled by program in the GPC.

For write functions, when less than a full record is changed, the record is normally read into the GPC, changed, and totally rewritten.

2.3.4.1.2 Read - The read function may occur in order to transfer into the GPC memory any of the types of data fields mentioned above from those locations identified by the access means. It is assumed that the DMA has a limit register that prevents spillover, when the read function tries to present too much data in the case where the quantity is unknown.

An important facet of the read function is that usually it is preferred to transfer only selected fields from the accessed records, to economize on GPC memory. This feature is provided by the control register which is loaded with field marks and transfer flags. (In the case of multi-block records this feature cannot be applied to more than one block per record.)

The read function, as defined, refers only to transfers from the data base on disc. Obviously there are many other data transfers from (and to) the AFP-2; e.g., it may do a transfer of the associative memory content to enable the GPC to do a programmed analysis of a search result.

2.3.4.1.3 Change Obsolete Flags - The function of changing obsolete flags usually, but not necessarily, accompanies other functions, and is executed concurrently. The use of obsolete flags is perhaps best illustrated in the updating of a dynamic file in response to a report. The report uniquely identifies a file entry. This entry is searched, read (in entirety), and obsoleted, by setting the obsolete flag. After being updated, the entry is written in the "first obsolete location" in a defined region on disc, thus reducing write access time. This operation is particularly effective where a batch of reports can be treated together, and the updated entries written in rapid sequence. With the write operation, the appropriate obsolete flags are reset.

2.3.4.1.4 Count - The count function is primarily used to provide statistics on compare flags after a search. It may be mechanized, as noted in the associative memory description (section 2.3.3.2), or it may exist implicitly through program in the GPC after associative memory content transfer. In situations where the count function is critical, there is also the possibility of doing it simultaneously in each word in the associative memory, provided the word length is expanded to include a count field. The technique of doing parallel by word arithmetic is adequately covered in Reference 4 and will not be repeated here.

2.3.4.1.5 Search and Set Compare Flags - The search and set compare flags function is the most important and complicated function of the AFP-2. As such, it is least precisely defined, in that many variations are possible. A fairly straightforward model is summarized here and variations are discussed later.

This function may be defined as follows: Set or leave compare flags (bits) "on" in the associative memory in those locations corresponding to non-obsolete blocks in the specified tracks which meet the search criteria contained in the key, mask, and control registers.

A simple model of the mechanization is presented here to assist in understanding the function and hardware. It is assumed that the disc memory bit duration (reciprocal of the bit rate) is long enough for many associative memory cycles.

(a) Associative Memory Role

During the reading of each block from all tracks in parallel, two things are occurring. First, the obsolete flags are being read into an assigned tag bit of each word of the associative memory. Second, key, mask and control information is being used with disc data in one or more tag bits to establish block comparison. (See Appendix A for tag bit logic.)

At the end of each block, during the spacer bits time, the results of the comparisons are "ANDED" with the obsolete flags and the result is stored in the bit of each word corresponding to the block just read. This process may start on any block so no access time is chargeable to it.

This process continues block after block for a disc revolution, after which all blocks (in our assumed model, 65, 536) have been completely searched and appropriate compare flags are set.

The search is conjunctive on the specified criteria in the key, mask, and control registers, i.e., all criteria must be met for a block to qualify. However, an effective disjuncton (logical "OR") of conjunctive searches can be attained by doing the necessary number of conjunctive searches and accumulating the compare flags.

(b) Key, Mask and Control Register Role

The key, mask and control registers (Figure 2.1) carry the criteria for a search and provide the following types of comparison on a field-by-field basis (a field is a number of contiguous 8-bit characters):

1. Equals
2. Greater than or equals
3. Less than or equals
4. Not equals
5. Bounded

For all but the bounded search, the key and mask registers contain a key and a mask. The **mask** actually in on a bit-by-bit basis, so individual bits can be used as flags in records.

For bounded search, the key and mask registers are used to contain the upper and lower bounds, respectively. As no bit masking is possible, the search applies to full 8-bit characters.

The control register has associated with it an 8-bit static register which is loaded each character time. This static register controls what happens during the subsequent character time. (Obviously, the control register content must lead the key and mask register content.) Individual bits have individual functions. One bit serves to **mark** off fields. Another masks data transfers. Three select among the search types.

2.3.4.2 Access Means

There are three access means characteristic to the data base. These are:

1. Fixed address
2. Obsolete flag
3. Compare flag

2.3.4.2.1 Fixed Address - Fixed address access implies that the GPC can specify any block in any track in any section of the disc memory as the starting location of a function. It also implies specification of the duration of the function, e.g., how many blocks are to be transferred. The transfer can go from track to track uninterrupted.

2.3.4.2.2 Obsolete Flag - Obsolete flag access is used only for the write function. It may be used for multi-block records, by applying it only to the first block of the record.

2.3.4.2.3 Compare Flag - Compare flag access is, of course, the primary retrieval mode of the AFP-2. From the structure of the word and bit counters of the associative memory, all blocks with compare flags set can be read through the transfer mask into the GPC in one efficient operation.

Compare flag access is primarily associated with the read function, although it is not unreasonable to associate it with other functions, such as "change obsolete status".

(a) GPC Control - Variations of the read by compare flag operation are possible as the result of differing levels of hardware complexity. The simplest in concept is where the compare flags in the associative memory are transferred into the GPC, which then issues a series of fixed address read operations to the disc memory. The fixed addresses are easily calculated from compare flag positions, but some assistance may be in order to help find set compare flags. In this case, after an "or" operation over the compare flags in the associative memory into a tag bit, this tag bit, transferred with the compare flags, will materially assist the GPC. If GPC time is available, the fixed address accesses to disc memory can be queued in time to correspond to the sequence on the disc. Thus, the transfer can be made time efficient, with a disc random access only to the head of the queue.

(b) AFP Control — A similar, even more efficient transfer can be made under control of the AFP if the associative memory has the priority circuit option. The AFP starts the transfer at the current position of the disc without a random access time lost, but with only a wait for the first qualifying block in any track.

This read function is mechanized as follows (see the example in Figure 2-2):

The bit (of each associative memory word), corresponding to the next disc sector to appear, is read into the detector plane. Then the priority circuit is activated and the first "one" is used to select the track to be read. The bit (compare flag), corresponding to the block being read is then reset to zero, and the process is repeated. In the next sector a qualifying block may be read from another track, which is all right, as track selection, presumably, is done at logic level.

With a sparse number of compare flags, by and large the read function will complete in one disc revolution. This assumes that no more than one track carries a qualifying block in any given sector. When this is not true, more than one revolution is taken in any possible read sequence.

2.3.5 Search and Retrieval Considerations

The following sections discuss the search and retrieval process in detail, going through it operation by operation. In some cases the logic mechanization is detailed. Timing is also discussed. The following sequence of operations takes place:

2.3.5.1 Region Selection

The search normally must be confined to a region on disc memory established by software to contain the file of interest. For instance, if a region is used to contain raw, unformatted data (perhaps linked to formatted files), a search might produce accidental makes or matches which would have to be identified and disposed of. The region selected is a set of contiguous tracks within a section of the disc memory.

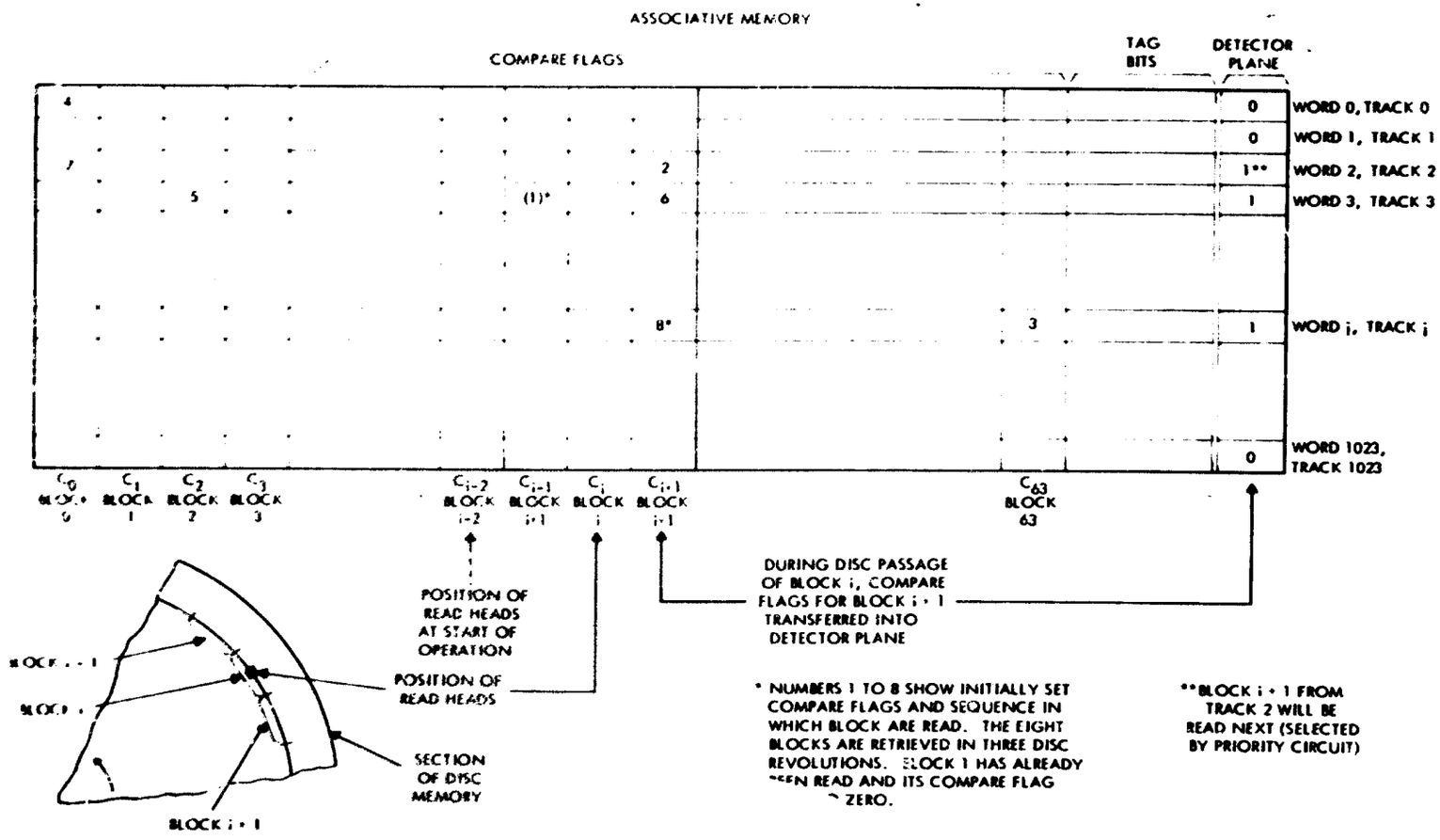


Figure 2-2 Example of Read by Compare Flag

2.3.5.1.1 Section Selection — It is assumed that there are several hardware-established sections in the disc memory. For example, there may be five sections, each containing 1024 tracks. Five-way electronic head selection for each read amplifier would then constitute section selection.

Another possible method of section selection would use a head positioner to physically move each head to one of several locations.

2.3.5.1.2 Track Selection - In the general case, a region is a subset of a disc memory section. There are a number of ways in which the region can be delimited. One way is to preload a dedicated tag bit in the associative memory, and use this in a way similar to the obsolete flag usage; i.e., both it and the obsolete flag are "anded" with the comparison tag bit to establish compare flags.

Another way is indicated in Figure 2-1. Through the use of lower limit (LL) and uper limit (UL) registers on the counter that distributes obsolete flags, a region can be defined simply by regarding all records outside the defined region as obsolete.

Both of the foregoing methods operate during the search. Another method can be used which follows the search; namely, to set unwanted words in the associative memory to zero, erasing the compare flags. Again the LL and UL registers can control this.

Whichever method is used, region selection is the first logical step in the search and retrieval process.

2.3.5.2 Load Key, Mask and Control Registers

As described in section 2.2.2.2, each file has a format directory which describes the entry format, i.e., the field locations and names, and permissive values. The query preprocessing uses this directory to establish the search criteria. These search criteria are then loaded into the key, mask, and control registers.

The roles of these registers in the search process are described in section 2.3.4.1.5 (b) and will not be repeated.

It is possible to load these registers in a number of ways. One is to generate their exact content in GPC and do a simple transfer. This tends to be cumbersome, and often minor changes to contents are desired between searches. Another method is to provide specific operations which load specified fields in any of the registers.

2.3.5.3 Initiate Search

The search process is initiated after region selection and loading the search criteria registers. If the search is a simple one, it is preceded by clearing all the compare flags in the associative memory to "zero". Alternatively, for a disjunctive search, the compare flags are not cleared, but accumulated.

The search begins as the first complete block begins to be read, regardless of which one it is, and continues for one revolution until all blocks are read.

The search completion is preferably signalled to the GPC via an interrupt.

2.3.5.4 Derive Search Result Statistics

Search result statistics may or may not be of interest to the GPC program. If they are, any of the methods described in section 2.3.4.1.4 may be used.

In particular, if detector plane statistics are available, a logical OR of the compare flags is put into the detector plane. Under the assumption of reasonably random (i.e., Poisson) compare flag distribution, this process is very accurate for small numbers of compare flags (10-20% of the tracks have matching records) and deteriorates mildly (63% valid) when there are as many matching records as tracks.

2.3.5.5 Read Search Result

The process of reading the matching records into the GPC is described in detail in section 2.3.4.2.3 under Compare Flag Access.

An option of the read operation is to set obsolete flags as blocks are read, section 2.3.4.1.3. This may be for the purpose of deleting blocks (and possibly writing them onto a historical tape) or for updating with replacement in the file by a write by obsolete flag access operation.

Another option of the read operation is to have each block accompanied by its disc fixed address. This allows replacement in exact locations in, say, a sequenced file. In fact, the block transfer can be totally masked out and only block addresses made to enter the GPC.

Formatting of the data transferred to the GPC is a function of its direct memory access (DMA) channel. Records should start in separate words, and it may even be desirable to put fields in separate words; i.e., each new field starts in a new word.

2.3.6 Multiblock Record Considerations

In instances where the data base contains files with one block records, files with two block records, etc., and perhaps some "raw" unformatted data, it is convenient to segregate the files by their record length. That is, one disc memory region will have only one block records but perhaps several files intermixed,and another only two block records. Normally, these regions will not overlap from one section of the disc to another (but they may; see section 2.3.8.2), and hence can be searched in one disc revolution.

Multiblock records are considered to have a header block and one or more trailer blocks. A given file always has a fixed record length, however. Blocks are tagged internally with not only their file identification, but their sequence number within the record. By including these tags as search criteria, any fields in the record can be searched on.

When the search fields for a given search are all within one block, a simple search suffices to establish compare flags. When the search fields come from different blocks, each block is treated in a separate search, establishing compare flags each time. The associative memory is then used to establish which records have compare flags in all required blocks. Those which do are left with their compare flags, adjusted, if required for the read process.

Multiblock records will have their header blocks in given sectors, i.e., two-block records always start in an even numbered sector. This allows a read process which is efficient as described in section 2.3.4.2.3. With only one control register, a field masked transfer is impractical, as each block of the record would have the same mask applied. It is practical, however, to mask block by block, as noted above, through operations on compare flags in the associative memory.

Multiblock write operations are straightforward using fixed addresses brought along in read operations. However, the multiblock write-first-obsolete-location operation requires additional hardware to identify header block locations. An extra disc track, called the header block flag track, is needed. This track is not a circulator like the obsolete flag track, but needs only a single head. It is written by the GPC when disc regions are assigned. With this refinement, many multiblock records can be written in a single write operation. The header block flag not only identifies the beginning of an

empty slot, in conjunction with the obsolete flag, but also the beginning of the next slot and hence the end of the empty slot.

Writing cannot be masked, even block by block; hence, full records must be assembled, in general, for write operations.

The comments on detector plane statistics relative to a count function remain valid.

2.3.7 File Maintenance Considerations

File maintenance involves adding, deleting, and changing records in the data base, in contrast with querying, which is concerned primarily with selective data retrieval. Changing records involves retrieving them and replacing them on disc memory. Not much can be done about the access time to retrieve a record, assuming it is done by a content search. As search and read are separate operations, the total retrieval time is a random access time plus a disc revolution, averaging 1 1/2 revolutions. The revolution corresponds to the search, and the random access to the read. Writing, however, is a rapid process, assuming the write in first obsolete location is used.

Note that this process does not compare unfavorably in access time with the technique of keeping a disc map in core memory, random accessing a record by fixed address, and then waiting a disc revolution to replace the record. Content retrieval, of course, obviates the need for a disc map in core memory.

Batching updates, assuming independent records are involved, becomes somewhat more efficient, as a single read operation, taking most of a disc revolution on the average, and a very fast writ first obsolete location are chargeable to the whole batch of updates. However, a separate search operation is required for each record. This suggests a new search-and-set compare flag function, namely, "search until a compare flag is found" which cuts average search time in half. The associative memory detector plane "match type" option is needed for this.

Using the write in first obsolete operation on a dynamic file, which is updated frequently, will have the effect of crowding entries in the lowest numbered tracks of the region containing it (because obsolete flags appear track number sequence). This reduces the validity of the count function

using the detector plane statistic option in the associative memory. However, it favors the scheme of counting in the GPC after associative memory transfer.

2.3.8 Tradeoffs and Options

This section is primarily a resume and discussion of tradeoffs and options previously mentioned with some additional ones introduced.

2.3.8.1 Block Length

As discussed in section 2.2.3.1, the block length is a somewhat arbitrary parameter which can be estimated to suit the application. Short blocks favor disc storage efficiency where formatted files are stored, but both require a larger associative memory for the larger number of compare flags per track, and increase the management cost of storing unformatted data. Also, short blocks increase the number of obsolete flag tracks and header block flag tracks.

Because of the way obsolete flags are loaded during the search, it is convenient to have an integral relationship between the number of tracks searched simultaneously and the number of bits per block. Of course, in static data base applications, where obsolete flags are of no utility, this constraint disappears.

2.3.8.2 Associative Memory

In section 2.3.3.2, the associative memory options of the priority circuit, or resolve operation, and detector plane statistics are mentioned. The resolve operation is particularly useful, as it enabled the very fast read by compare flag operation, described in section 2.3.4.2.3 (b).

If structured in a particular way, the resolve operation can be used to obtain detector plane statistics through an iterative process. Successive resolve operations are performed and the selected detector plane bit is set to zero. A count of these operations is the required statistic.

Other means of generating detector plane statics have a usefulness highly dependent on the application.

The size of the associative memory compare flag region directly determines the number of blocks that can be searched in one disc revolution. Both associative memory cost and the fact that in most applications there are many files, only one of which is searched at a time, dictate the search of only a subset of the disc tracks at a time.

The tag bits required vary with the application. A total of four are needed in the presently assumed structure. One is devoted to the obsolete flags and is loaded one bit at a time as the flags are read. Three are needed to handle all of the types of searches assumed. The equals, greater than or equals, less than or equals, and not equals searches require two tag bits, one active during each searched field and the other to remember the result from field to field. However, the bounded search requires the three dedicated tag bits. The search operation, exclusive of the field demarcation is described in the RADC report[4] and so is summarized in Appendix A.

The search, insofar described, is a conjunctive one, in that all qualifications in the search criteria must be met. Disjunctive searches, then, become merely sequences of separate conjunctive searches. It is useful to consider the complementary structure for some applications. One such application exists in context of the DDC (formerly ASTIA) retrieval problem, discussed in Appendix B.

Another relationship of the associative memory and the disc memory is worth consideration; namely, where there is one large homogeneous file occupying most of the disc, and this file seldom, if ever, changes. Obsolete flags are not necessary, as it is assumed that file maintenance consists almost exclusively of adding new items. Also, it may be desirable to search many or all disc memory sections, establishing compare flags, in a single (multi-revolution) operation. (When combined with the disjunctive-then-conjunctive search, this becomes the DDC retrieval problem.)

As the associative memory cannot hold compare flags for more than one section, it is necessary to use disc memory for compare flag storage. As a consequence, only one bit of compare flag storage is needed in the associative memory (per word). These bits are written serially, as the search proceeds, into a track structured the same as the obsolete flag track described previously. Note that compare flag statistics are easy to obtain by a simple serial counter.

2.3.9 Conclusions

This section has discussed in detail the design of an associative file processor, which can perform the parallel file processing described in section 2.2. It not only achieves the high search speed ascribed to parallel processing, but is efficient in the broad spectrum of operations required of a dynamic data base storage and retrieval device.

The AFP-2 design is described in a somewhat generalized manner, pointing out a number of the speed/hardware tradeoffs and function options, because it is recognized that any actual design will be highly application oriented. In spite of the hardware variations, the nucleus of the design goal is retained, namely, the ability to do a rapid, highly parallel content search of files in a data base.

2.4 A QUERY LANGUAGE FOR USE WITH AFP

This section presents a query language suitable for use with the AFP when operating in an information retrieval environment. The language is that developed for the 473L Command and Control System.[1] Identification of a suitable AFP query language accomplishes the following objectives:

1) The language aids in delineating expected areas of application for the machine.
2) AFP hardware was utilized for translation from the query language to machine language.

The 473L query language was chosen as the AFP language because it is operational, well documented, and its intended usage area closely parallels that of the AFP. Librascope, having developed the 473L hardware, is familiar with the language. The original QL was implemented on an IBM 1401 system, about four years ago, during the "Operational Training Capability" (OTC) phase. In June 1964, the OTC system was upgraded to an IBM 1410 system. The second generation of 473L was implemented in the third quarter of 1964, with the installation of a Librascope L-3055 Data Processing System. This was the second phase of incremental growth. It represented the "Initial Operational Capability" (IOC) stage. Development of this system is now in the third phase, the "Complete Operational Capability" (COC), which was initiated in mid-1965.

During each of these operational phases, 473L-QL and its users were under constant evaluation. The language was thereby refined and improved at each stage of development. The fact that 473L-QL is operationally proven was an important reason for its choice as a model for the present study.

The QL is composed of two major sections, vocabulary and grammar. The vocabulary consists of two parts, fixed and dynamic. Content of the fixed part is a set of items which describe and control the search and retrieval processes. There are three types of items: words, symbols and punctuation.

The dynamic part of the vocabulary contains words which describe the data base information to be processed. The principal word type is "attribute name". This part of the language is dynamic in nature since its content is variable. This occurs when the data base content is altered, resulting from files, attribute names and values being added, deleted or modified.

Data files to be used with the AFP are in fixed record format. Why this structure has been chosen is explained in Section 2.2, with regard to parallel processing. As a consequence of this fixed format, a constraint is placed upon the file storage of attribute values associated with each attribute name. When it is possible for a variable number of values to be present under one attribute name, the format must be structured so as to contain the maximum number of values to be allowed. For some attribute names, this structure will contain some empty, or blank, value-spaces. This cannot be avoided under the fixed-record format.

The QL grammar is implemented by syntax and punctuation. The syntax, ie., proper arrangement of words into a meaningful query, is set into a format composed of items drawn from the fixed part of the vocabularly, as described previously in this subsection.

With regards to using the QL to form queries, punctuation marks appear as they would in a normal English sentence. The statement format approximates that which is used in English grammar. Use of special symbols has been almost eliminated. These factors have resulted in a close approximation to the ideal man-machine relationship.

The seven basic statement elements of 473L AL are now described. In order of their use, these elements are:

Program Indicator (1)

File Indicator (2)

Qualifier Conjunction (3)

Qualifier (4)

Output Conjunction (5)

Output Director (6)

Output Selector (7)

In certain QL statements (queries) some of these elements are omitted. In others, elements appear more than once.

1) Program Indicator

This is the initial word of the QL statement. It directs the control program to use the QL program. It also provides a logical English language beginning for the statement.

Example: Retrieve

2) File Indicator

This identifies the file from which data are to be retrieved. It always follows the program indicator.

Example: Aircraft (File name)

3) Qualifier Conjunction

The specific word "with" follows the file indicator. It serves as the conjunction between the file indicator and the qualifier. Also, it makes a more readable statement and eliminates the need for punctuation to identify the following qualifier.

The QC: with

4) Qualifier

This is the element of the statement which describes the specific nature of the data to be retrieved. A qualifier consists of a set

of one or more modifiers, each of which is normally composed of an attribute, a comparator, and a value. An attribute is a characteristic of the file; a value is one of the states an attribute may assume; and a comparator defines the logical or mathematical relationship between the attribute and the value. RUNWAY LENGTH is an attribute of the sample aircraft file, and 5000 feet could be a value for RUNWAY LENGTH. The expression "RUNWAY LENGTH>5000" is therefore a valid modifier. Another modifier could be "COMMAND = TAC". Placing these two together as "COMMAND = TAC, RUNWAY LENGTH>5000" forms a modifier set that describes certain entries in the file more specifically. The modifiers in a set are separated by commas and are logically additive; that is, an entry must meet the requirements of all the modifiers in a set to qualify. A simple qualifier contains only one modifier set. A compound qualifier may be constructed by combining several alternative modifier sets. This could be "COMMAND = SAC, AND ACFT POS>10; OR COMMAND = TAC, AND RUNWAY LENGTH>5000." The semicolon (;) defines the end of one modifier set and the beginning of the next. It also specifies a logical OR relationship between the sets. Data may qualify by meeting either of the modifier set's criteria, in this case. If the (above) semicolon is replaced by a comma, file data qualifies only if it meets the criteria in both modifier sets.

5) Output Conjunction

The specific word "then" always follows the qualifier and separates it from the output director. The conjunction also makes the statement more readable.

The OC: then

6) Output Director

This specifies the output device and the format in which the retrieved data are to be presented.

Example: Print, H (horizontal)

7) Output Selector

The last part of the QL statement is the output selector. It contains the attribute names which are to be output with their associated values. Also specified, if necessary, if the detail arrangement of this output, within the format given in the output director.

Example: Command, Afld Name, Acft. Pos, Runway Length

Query statements are terminated by the "end of message" symbol, ¬ .

Beyond the normal use of the seven statement elements just described, there are optional features available to the user. They are used in conjunction with the Qualifier (4), Output Director (6) and Output Selector (7). These options are described as follows:

a) SAVE (with Output Selector)

This specifies that the input query statement is to be saved, ie., stored in the SAVE table, for later use.

b) REMARKS (with Output Selector)

Any comments (free text) which the user wants to have included with his statement are appended at this point. This type of information is stored in the second section of the files as previously described.

c) TITLE (with Qualifier and/or Output Selector)

This option allows input and/or retrieved data to be titled. For its identification, an asterisk must be

placed at each end of the title, e.g.,

TITLE (Content)

d) SORT (with Output Selector)

If the output information is to be presented as sorted data, the mnemonics, INCR and/or DECR are used. If the data is to be forward sorted, ie., A to Z and zero to nine, INCR, ie., increase, is used. For the oppositely ordered sort, DECR, ie., decrease, is inserted.

Sorting can be performed on a number of different attributes in one statement. The attribute listed first represents the values to be used in the primary sort. Succeeding attributes, in the order listed, represent subsequent levels of sort.

e) UPDATE (with Output Director)

If it is required to update data files with new information, the word, UPDATE, is applied to the attributes and values to be used.

f) RETAIN (with Output Director)

This director is similar to the previous one, UPDATE. It specifies the output attribute values which are to be saved in tabular form for later use.

All QL statement information, representing both basic element and optional items is set into a specified format. In addition to the basic elements and options discussed previsouly, the QL contains a few specific functions. Their use is indicated by insertion of function mnemonics into the qualifier or output selector section within the QL statement format. These functions

provide a user with the ability to qualify or to generate and select certain data, using as control criteria, information which is not explicitly stored in the data base files. The functions which currently are available are defined as follows:

Great Circle Distance (GCD) - computes the GCD between two geographic points on the earth's surface.

Sum (SUM) - accumulates the sum of values of a particular attribute.

New File (COMBINE) - extracts information from several data files and generates a new file to be used, as desired, in subsequent query statements.

By way of illustration, one of these functions, SUM, is now described.

The SUM function can be performed on any numeric attribute values within the query statement. It is used in either the Qualifier (4) or Output Selector (7) part of a query statement.

The general form, in the Qualifier part of a statement, is expressed by using the term, SUM, followed by: the attributes which control execution of the function, attributes to be summed, and their respective comparators and values.

A typical example is: SUM BY COMMAND (ACFT RDY>14).

If a sum, as described above, is desired as output information, the Output Selector part of a statement is used. Only the term, SUM, is required for this action.

If attributes which are to be summed for output do not appear in the Qualifier part of a statement, a particular Output Selector format must be used. This

form consists of only control attributes and the attributes to be summed. A typical example is: SUM BY COMMAND (ACFT RDY, ACFT POS).

One additional feature of the QL, not previously discussed, provides for the generation of complex queries. A complex query is composed of more than one subordinate query. Two successive subordinate queries are separated by a colon (:); they can address the same or different data files. In operation, data retrieved in response to a particular subordinate query are used in succeeding subordinate queries as attribute values. In this way, the user is provided with a means by which he can use data, from certain files, to identify and select data from other files.

A complex query example is given:

In English text, as a user would present it, the example statement is,

"Retrieve each SAC airfield name and the number of crews formed, for those bases having at least as many crews formed as there are "COMBAT READY" aircraft at the Offutt SAC base."

Using the foregoing plain test, the query in 473L language is as follows:

```
RETRIEVE (ACFT WITH) COMMAND = SAC, AFLD NAME = OFUTT THEN
RETAIN ACFT RDY:  RETRIEVE PERS WITH COMMAND = SAC, CREWS
FMD ≥ (ACFT RDY, CREWS FMD, OR) THEN PRINT, H* OFFUTT, ACFT
RDY * ACFT RDY * AFLD NAME, CREWS FMD * AFLD NAME, CREWS
FMD¬
```

Explanatory notes, on this query, are presented below:

ACFT, the Aircraft Data File

The colon (:) separates the two SQ's of this complex query

PERS, the Personnel Data File

(ACFT RDY, CREWS FMD, OR), denotes comparison of each

"CREWS FMD" value with "ACFT RDY." The "OR" specifies that each successful comparison (ie., on≥) is a valid result; meeting the user's requirement.

REFERENCES - CHAPTER 2

1) Auerbach Corp. Analysis of Small Associative Memories for Data Storage and Retrieval Systems Contr. AF 30(602)3564, Sept. 1965

2) Barlow, A.E., and Cease, D.R., Command and Control System Query Language, Headquarters, United States Air Force presented at the Second Congress in The Information System Sciences, 1965

3) Campi, A.V. et al Content Addressable Memory System Concepts IEEE Trans on Aerospace and Electronic Systems Vol. AES 1, p. 168 (1965)

4) Fuller, R.H. et al, Study of Associative Processing Techniques RADC-TR-65-210 August 1965 (DDC No. AD 621 516)

5) IBM Federal Systems Div., Sea Surveillance Data Base Representation As Test Vehicle prepared for ONR June 1964

6) Kochen, M., Some Problems in Information Science, The Scarecrow Press, Inc. New York, 1965

7) Love, H.H. and Savitt, D.A., Associative Processing Techniques Study RACD-TR-65-32 April 1965

3.0 ASSOCIATIVE SOLUTION OF NETWORK FLOW PROBLEMS

3.1 INTRODUCTION

In this section, an associative parallel processor (APP) is structured and programmed for solution of network flow problems, typified by resource allocation problems and commodity transportation problems, widely discussed in operations research literature[1-12]. The structure and command set for the APP are very similar to those previously developed by Librascope under RADC contract AF 30-(602)-3371 for picture processing applications[15,16] The studies reported in this and in the following chapter are intended to extend the range of identified applications for which the associative parallel processor can be efficiently employed when compared to serial processors, and to explore variations in machine organization and command set occasioned by changed problem environments.

Each network flow problem, herein considered, is a variant of the general Hitchcock-Koopmans Transportation problem[1,2] and is solved by a variant of the "Hungarian Assignment Method" due to Kuhn[5,6]. In the remainder of this section, we describe the network flow problems to be solved, and show how weapon assignment tasks can be couched within these models. We describe the structure and command set for the APP used to solve these problems, noting organizational variations from that APP developed for picture processing. We then program the APP for solution of three distinct forms of the transportation problem and compare associative solution times to times for solution of these problems on conventional serial machines. It is shown that associative parallel solution of each transportation problem investigated is one-to-three orders of magnitude faster than conventional processing, dependent on network size. An important consequence of this result is the ability of associative parallel processors to solve complex weapon assignment tasks in real time. A summary, and conclusions reached, are included in subsection 3.4.

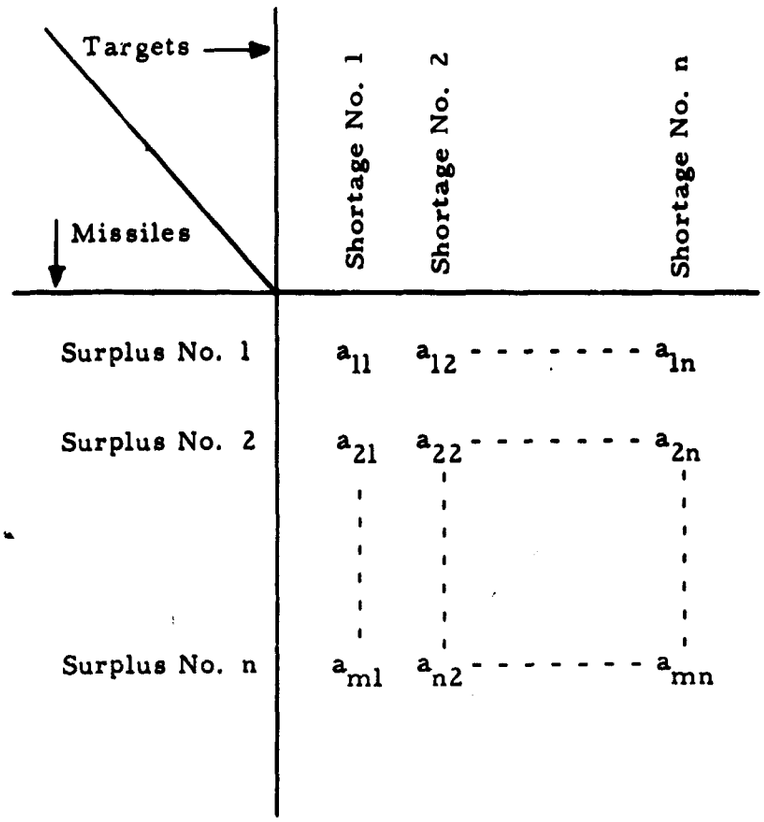
The general Hitchcock-Koopmans transportation problem was first formulated by F. L. Hitchcock in 1941[1], and independently, during World War II, by T. C. Koopmans[2]. Both men suggested a procedure for solving this problem similar to the linear programming simplex method[3]. In 1951, Dantzig[4] developed a simplified form of the simplex method for solving this problem. This method was popularly employed until 1955, when H. Kuhn[5, 6] proposed a method based on a combinatorial procedure. Kuhn termed this method "The Hungarian Assignment Method" after the Hungarian mathematician Egervary[7], who provided the essential feature of the proof for a theorem of Konig[8] concerning linear graphs. Kuhn used this proof as the basis for his algorithm which provided a simpler method than Danzig's simplified simplex method for solving the assignment problem. Later in that same year, Ford and Falkerson[9] discovered an algorithm similar to Kuhn's which could be used to solve the more general transportation problem. A concise statement of Kuhn's, and Ford's and Falkerson's algorithms can be found in an article by Munkres[10].

There are two variations on the Hitchcock-Koopmans transportation problem which have been termed:

1) The assignment problem

2) The transportation problem

As will become evident, the assignment problem is only a special case of the more general transportation problem. In each case, there is a rating matrix (a_{ij}), as illustrated in Figure 3-1, where the rows of the matrix correspond to initial surpluses of materials, eg., missiles, and the columns correspond to initial shortages, eg., targets. The elements, a_{ij}, of the rating matrix represent costs or values associated with sending each of the surpluses to each of the shortages. In Hitchcock's original problem, the row designators represented warehouses with inventory surpluses and the column designators represented stores with inventory requirements. The given rating matrix represented costs associated with shipping the surpluses from each of the



- Given a rating matrix (a_{ij})

  Find an assignment matrix (x_{ij})

  To maximize/minimize the object function:

$$\left[\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ij} \right]$$

Figure 3-1 Example of Rating Matrix

stores. The problem was to find the assignment of the inventory from each of the warehouses to each of the stores, as repreeented by an assignment matrix (x_{ij}), such that the total shipping cost was minimized, i.e.,

$$\text{Minimum} \left[\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} \, x_{ij} \right]$$

In the case of a weapons assignment problem, the rating matrix can be thought of as a value matrix, where:

$$a_{ij} = W_j \, D_{ij} \, P_{ij}$$

and

W_j = Worth or value assigned for each target "j"

D_{ij} = Destruction capability for each target "j" by each missile "i"

P_{ij} = Probability of each missile "i" reaching each target "j"

a_{ij} = Expected worth of each missile "i" for each target "j"

Here, the problem is to determine the assignment matrix (x_{ij}), so that the total expected value is maximized, ie.,

$$\text{Maximum} \left[\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} \, x_{ij} \right]$$

For the so-called "Assignment problem", each of the row designators contains <u>one and only one</u> unit of surplus and each of the column designators has a shortage of <u>one and only one</u> unit. Accordingly, the rating and assignment matrices are of dimension n × n where

$$x_{ij} = 0, 1$$

In the so-called "Transportation Problem", each of the row designators has <u>one or more</u> surplus units and each column designator has a shortage of <u>one or more</u> units where the total surplus units is equal in number to the total shortage units. Here the rating and assignment matrices are of dimension m × n.

Three variations on the network flow problem were considered, namely:

1) Binary assignment problem

2) General assignment problem

3) Transportation problem

The second and third variations above, were previously explained. The binary assignment problem is a simplified version of the general assignment problem where the rating matrix is binary, i.e., consists of only zero's and one's. In other words, each missile either can or cannot be assigned to a target. This problem is simply one of finding any feasible solution for the assignment matrix x_{ij}, since all feasible solutions are equally optimum. This problem was selected for consideration because its solution using two serial processors, the L-3055 and the AN/FSQ 31(V), was available in a Librascope report including timing data. The timing data allows a convenient comparison between the APP and a serial processor.

3.2 THE ASSOCIATIVE PARALLEL PROCESSOR

3.2.1 The Structure of the Processor

Figure 3-2 is a block diagram of the APP, structured to solve the weapons assignment problem. This APP is similar to the one structured for pattern recognition described in a prior RADC report[15]. Elements of the APP in Figure 3-2 identical to the previous APP are:

1) Random Access Control Memory - a small memory used to store the instructions.

2) Central Control - used to interpret the instructions in the random access control memory and execute control.

3) Associative Array - some of the fields in the array are used to store data while other fields are used for tagging purposes.

4) Data Drivers and Control - used to exchange data from the data register into the data fields of the associative array and vice versa and to issue pulses to interrogate the data fields of the associative array under a specified search criterion. Particular fields of the data fields in the

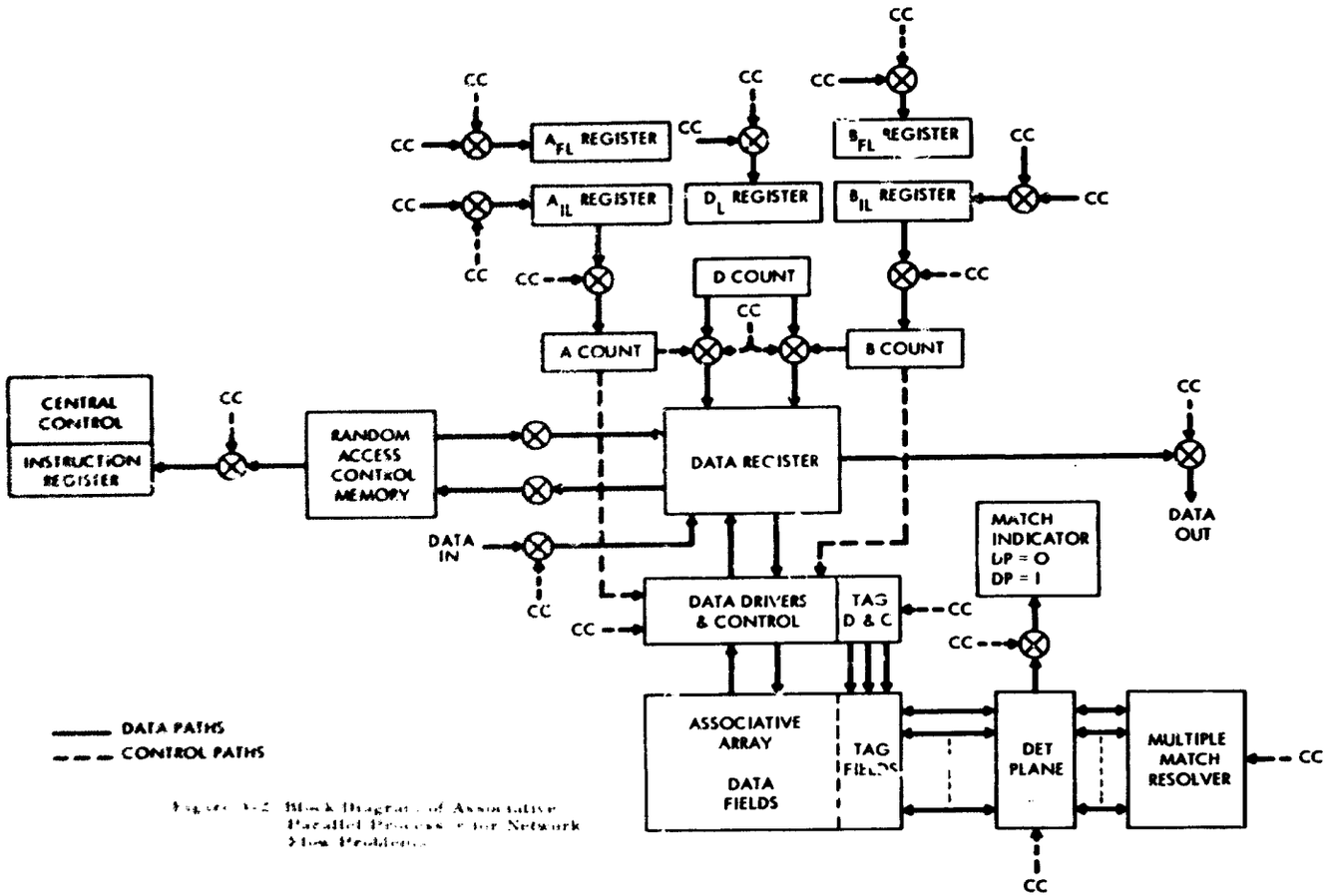


Figure 3-2 Block Diagram of Associative Parallel Processor for Network Flow Problems

associative array are selected under the control of the "A" and "B" counters.

5) Tag Drivers and Control - used to execute pulses to interrogate the tag fields of the associative array under a search criteria specified in the instructions.

6) Data Register (DR) - used to store data for reading into or out of the data fields of the associative array or the random access control memory. All input and output data are processed through the data register. It is also used to store a key for searching the data fields in the associative array.

7) Detector Plane (DP) - used to indicate a match in the associative array and to multi-write into the associative array for matched words.

8) Match Indicator DP = 0 - used to indicate when all detector plane elements are zero (no matches) after an associative search.

9) "A" Counter - used to specify and limit particular data fields for a search operation.

10) "$A"_{FL}$ Register - used to store the final count limit for the "A" counter.

11) "B" Counter - used to specify and limit particular data fields for a search operation.

12) "$B"_{FL}$ Register - used to store the final count limit for the "B" counter.

New processor elements useful in solution of network flow problems are:

13) "A_{IL}" Register - used to store the initial limit. After the final limit is reached on the "A" counter the next increment or decrement count indication will automatically reset the "A" counter to the initial limit.

14) "B_{IL}" Register - same as "A_{IL}" except for "B" counter

15) "D" Counter - used to store a value which can be transferred to the data register into fields as specified by a limit count of either the "A" counter or "B" counter. It is incremented under program control and initially starts with a count of one.

16) "D_L" Register - used to store the limit value for the "D" counter. When the limit value is reached, the "D" counter is automatically reset to one.

17) Match Indicator DP=1 - used to indicate when one and only one detector plane element remains in a one state after a search (a single match).

18) Multiple Match Resolver - when two or more detector plane elements remain in the one state after a search this network selects one matched element from the two or more.

The purpose of each of the elements in the structure will become more obvious after reviewing a description of the command set in the next subsection.

3.2.2 The Command Set

The command or instruction set can be partitioned into six different types, namely:

1) Associative Command

2) Counter Loading Commands

   a) Digit driver control counters

      i) Set $A_{FL} = x_{FL}$, $A_{IL} = x_{IL}$

      ii) Set $B_{FL} = y_{FL}$, $A_{IL} = y_{IL}$

   b) Data Counter

      i) Set $D_L = z_L$

3) Branching Commands

   a) Jump (unconditional) to _____

   b) IF "A" counter = initial limit, jump to _____

   c) IF "A" counter ≠ initial limit, jump to _____

   d) IF "B" counter = initial limit, jump to _____

   e) IF "B" counter ≠ initial limit, jump to _____

   f) IF "D" counter = 1, jump to _____

   g) IF "D" counter ≠ 1, jump to _____

h) IF DP (detector plane) = 0, jump to _____

i) IF DP ≠ 0, jump to _____

j) IF DP = 1, jump to _____

k) IF DP ≠ 1, jump to _____

4) Data Transfer Commands

a) Transfer matched word into data register

b) Transfer data registers into selected word

c) Transfer data register into RAM word _____

d) Transfer RAM word _____ into DR.

e) Transfer "D" counter into data register under A counter control

f) Transfer "D" counter into data register under "B" counter control

5) Multiple Match Command

a) Clear and select first match

b) Select next match

6) I/O Commands

a) Read input data into data register

b) Read data register into output data

c) Stop

d) Begin

The associative command is the most complex of the group and is further described in Figure 3-3. The first bit in the instruction indicates whether the command is associative or not. The second bit indicates whether or not the detector plane is set to one before the associative operation is executed.

The third through eighth bits concern search and multiwriting operations on the data fields under control of the "A" counter. The third bit indicates whether or not there is a search, the fourth and fifth bits specify the search criterion to be a zero, one, the contents of the data register, or the complement of the data register contents.

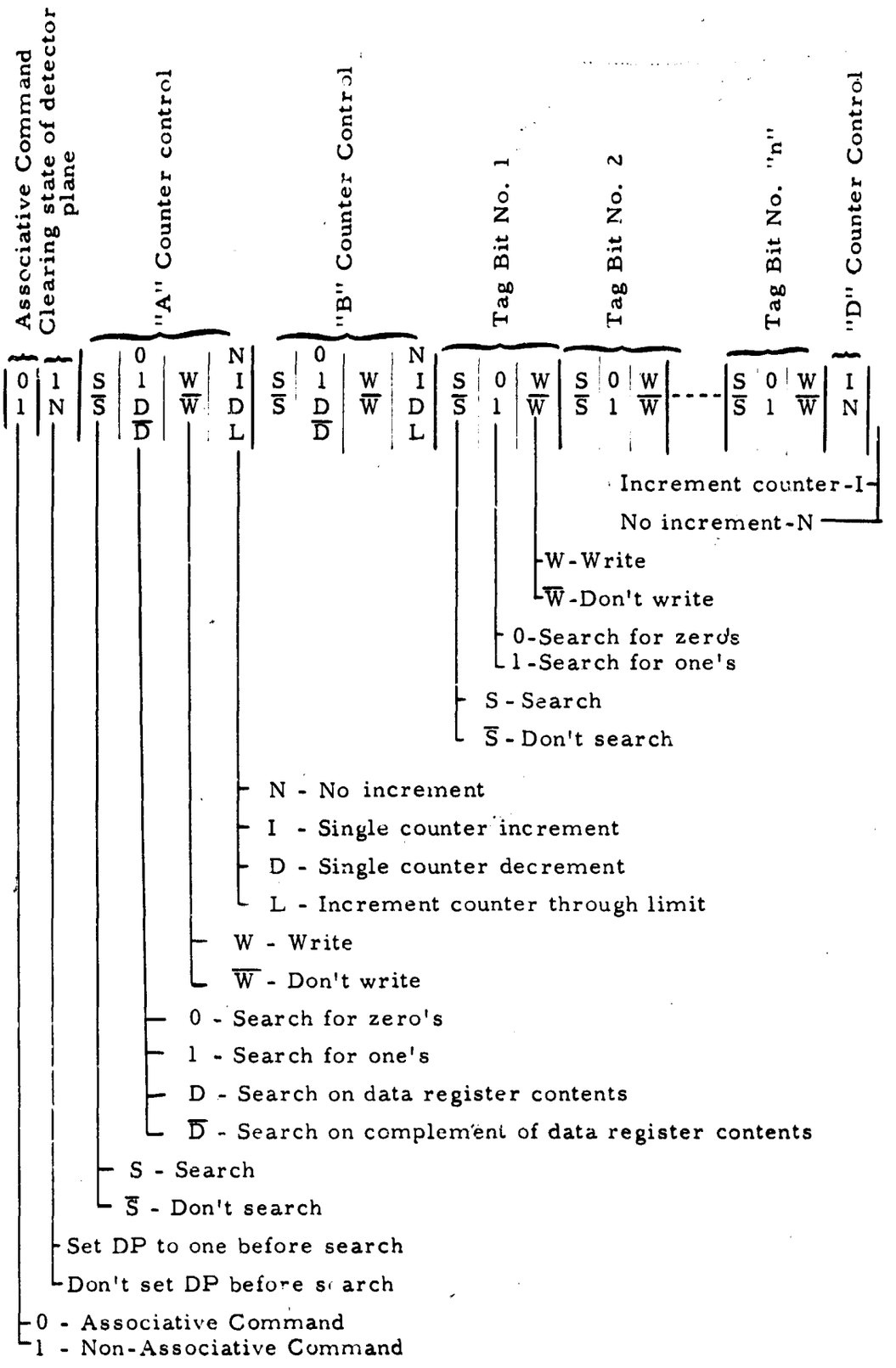


Figure 3-3 Format for Associative Command

The sixth bit indicates whether or not multiwrite is performed. If multiwrite is specified, then the logical complement of the search criterion is rewritten. The seventh and eighth bits indicate the increment or decrement state of the "A" counter, where "N" means no increment, "I" means increment (single), "D" means decrement (single) and L means increment through limit.

The ninth through fourteenth bit apply to the "B" counter and have the same specifications as the third through eight bits do for the "A" counter.

The remainder of the associative instruction word, except the final bit, is partitioned into fields of three bits, each of which specifies search and multiwrite operations on the tag fields of the associative array. The first field bit, e.g., bit 15, indicates whether the related tag bit is searched or not; the next bit, e.g., bit 16, indicates whether or not the search is on a zero or one; the final bit, e.g., bit 17, indicates whether or not the multiwrite operation is performed. Again, if multiwrite is specified the logical complement of the search criterion is rewritten.

The final bit concerns the increment state of the "D" counter where "N" represents no increment and "I" represents increment.

3.2.3 Timing Assumption on Command Set

The timing assumptions are based on a woven plated-wire associative memory described in Appendix C. This woven plated-wire associative memory has a digit search time of 50 nanoseconds and a multiwrite time for all like zero's or one's of 100 nanoseconds. Accordingly, as will be shown later, since no more than four tag bits are required for any one associative search in any of these three problems, it is assumed that the basic associative command can be performed in 500 nanoseconds with the exception of a limit search. In the case of a limit search, it is assumed that an additional time of 50 nanoseconds per digit through the limit will be required for the associative command cycle.

It is assumed, for compatibility reasons, that the random access control memory has an access time of 250 nanoseconds, and the instruction decoder has single instruction look-ahead and can execute a non-associative command, with three exceptions, in 250 nanoseconds. The three exceptions are:

1) Clear and select 1st match

2) Transfer matched word into DR

3) Transfer DR into selected word

In each of these three cases, it is assumed that 500 nanoseconds will be required to execute the command. The 500 nanosecond assumption can be justified in these three cases by referring to the description of the plated wire memory in Appendix C.

3.3 ALGORITHMS FOR SOLUTION OF NETWORK FLOW PROBLEM

Subsection 3.3 is organized into four lower ordered subsections. Subsections 3.3.2, 3.3.3 and 3.3.4 contain a description of the detail algorithms for the binary assignment problem, general assignment problem and transportation problem, respectively. However, before launching into a description of these detailed algorithms, it was felt that a simple example to demonstrate the types of operations required of the APP by the algorithms would be helpful. Accordingly, in subsection 3.3.1 an example, taken from the general assignment problem type, is presented. It is hoped that this example will serve to clarify the detailed algorithms presented in subsections 3.3.2, 3.3.3 and 3.3.4.

3.3.1 Example to Demonstrate Processing Operations

Before proceeding to the detailed algorithms presented in subsections 3.3.2, 3.3.3 and 3.3.4, it is worthwhile to present a numerical example to demonstrate the types of processing operations required and to show the utility of the APP in performing these operations. The particular example is selected from the general assignment problem type. The rating matrix for the example is presented in Figure 3.4, A). The object function will be maximized.

The Hungarian Assignment Method is based on the fact that a rating matrix (Figure 3.1) may be transformed by subtracting a constant from any row or column, without altering the optimum assignment matrix for the rating matrix. Through a series of such transformations, a rating matrix is altered to have a number of "independent" zeros equal to the order of the matrix. A set of zeros is independent if no two zeros lie in the same row or in the same column. The optimum assignment matrix is a binary matrix of the same order as the given rating matrix. It has ones at elements corresponding to independent zeros of the transformed rating matrix and zeros elsewhere.

A) Pick maximum in rows 1-6

RATING - MATRIX

6	~~10~~	3	0	0
~~15~~	8	0	14	12
0	2	~~8~~	7	7
~~7~~	0	3	4	0
5	2	2	~~7~~	0

B) Obtain at least one zero in each column 7-13

4	0	7	10	10
0	7	15	1	3
8	6	0	1	~~1~~
0	7	4	3	7
2	5	5	0	7

C) Initial independent zero assignment 14-22

4	0*	7	10	9
0*	7	15	1	2
8	6	0*	1	0
0	7	4	3	6
2	5	5	0*	6

D) Step No. 1 - Cost Matrix Test 23-26

4	0*	7	10	9
0*	7	15	1	2
8	6	0*	1	0'
0	7	4	3	6
2	5	5	0*	6

E) 27, 28 and 23, 24

4	0*	7	10	9
0*	7	15	1	2
8	6	0*	1	0'
0	7	4	3	6
2	5	5	0*	6

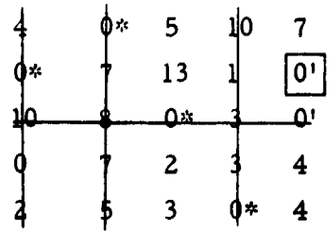
F) Step 3 - Cost Matrix Adjustment 39-41

4	0*	5	10	7
0*	7	13	1	0
10	8	0*	3	0'
0	7	2	3	4
2	5	3	0*	4

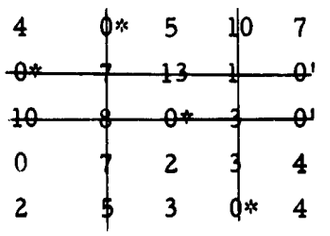
Figure 3-4 Processing Operations in Solving General Assignment Problem

RATING - MATRIX

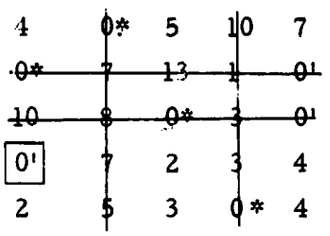
G) Step 1 - Cost Matrix Test 23-26



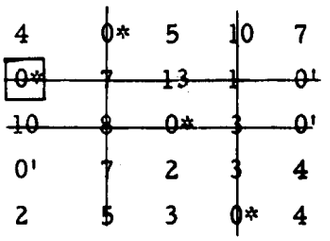
H) 27, 28



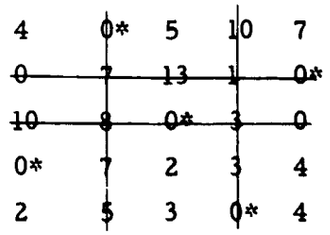
I) Step 1 - Cost Matrix Test 23-26



J) Step 2 - Assignment Adjustment 29-31



K) 32-34



L) 35-36

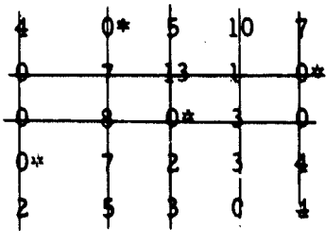


Figure 3-4 (continued)

The terminology used in describing the algorithm is due to Munkres[10] and is presented below:

1) Covered row or column in matrix - A row or column containing a starred zero in a transformed rating matrix.

2) Starred zero - Designated by a star (*) and used to indicate a trial assignment of an independent zero.

3) Primed zero in matrix - Designated by a prime (') and used to indicate a non-covered zero which becomes a candidate for starring during the course of the algorithm.

Addition designators used to point out particular elements in the matrix in the example of Figure 3-4 are:

4) An element with a single line through it - Used to designate the maximum value in a row or column.

5) An element with a square around it - Used to point out a particular element, as defined in the text, for the convenience of the reader.

The algorithm is illustrated in Figure 3-5. Parts A through L of Figure 3-4 represent the transformations on the rating matrix according to the algorithm of Figure 3-5 in solving this problem. Each alteration is discussed in turn.

A) The element(s) with maximum value in each row is (are) determined and the rating matrix value is subtracted from this value for each element. These operations correspond to instructions 1 through 6 in Figure 3-5. The result is shown in Figure 3-4B. These operations transform the maximizing problem into a minimizing problem. The APP can globally select the maximum element within a row in a single operation and can also globally perform the subtraction in a single operation. A serial processor would have to test each element in a row to determine the maximum as well as perform the subtraction for each element.

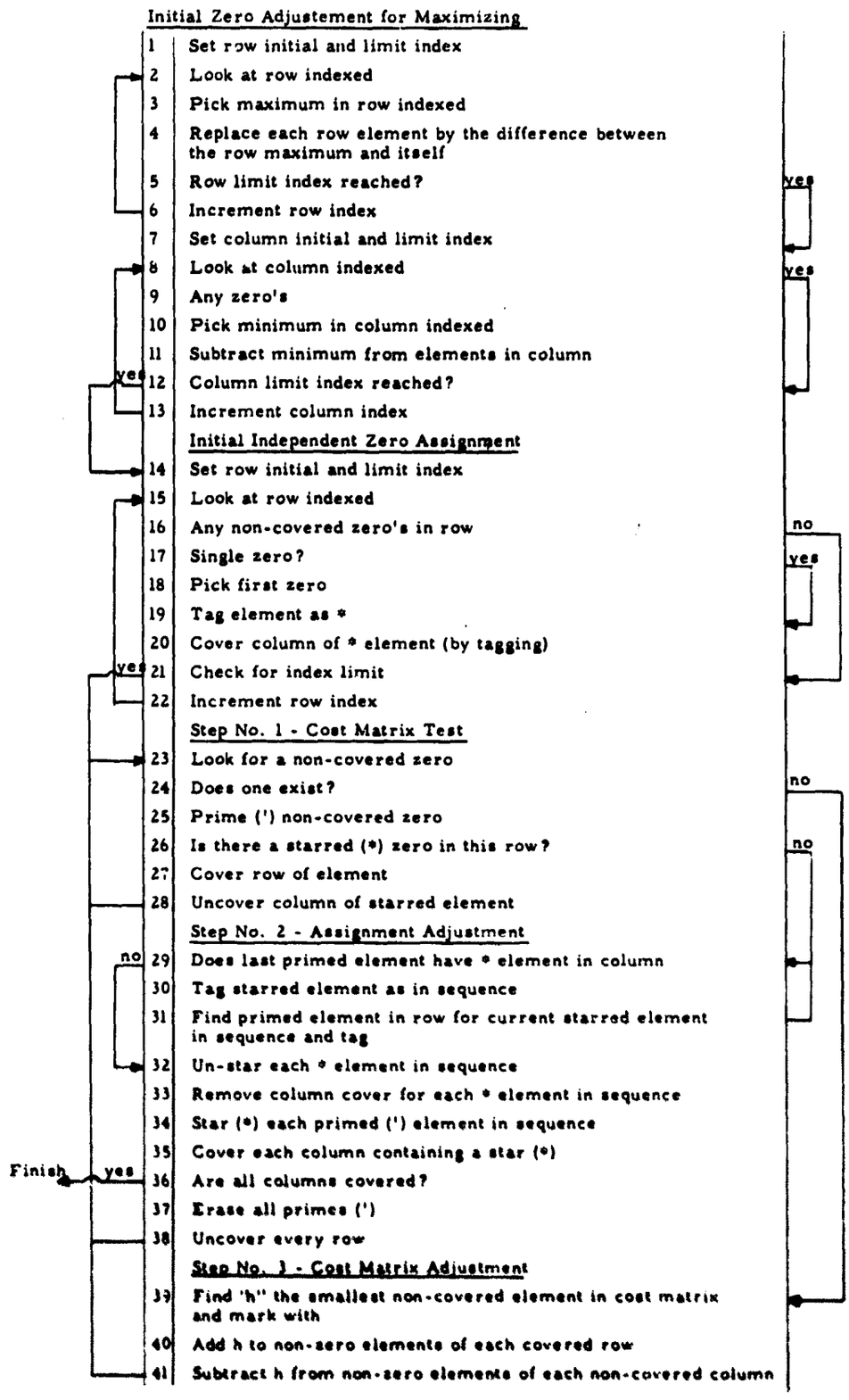


Figure 3-5 General Assignment Problem Macro Program

B) Each column is searched in order to guarantee that it has at least one zero. When a column is found without at least one zero, the minimum value for that column is selected and subtracted from all elements in the column. These operations correspond to instructions 7 through 13 in Figure 3-5. Note that only column 5 does not have a zero, and the minimum value of one is subtracted from each element as it is shown in Figure 3-4 C). Again, the determination of no zero's in a column, the selection of the minimum element and the subtraction, can be performed globally in the APP, whereas these operations would have to be performed sequentially in a serial processor.

C) Each row is searched for at least one non-covered zero. If one or more are available, one is selected and starred (*) and each element in the column for the starred element is covered by a line. These operations are shown in Figure 3-4 C) and described by instructions 14 through 22 in Figure 3-5. Again, these operations can be performed globally by row in an APP, whereas a sequential search would be required by a serial processor.

D' Here, the rating matrix is searched to determine whether or not a non-covered zero exists. If one does exist, as it does in Figure 3-4 D), it is primed as in D) which corresponds to instructions 23 through 26 in Figure 3-5.

Note that with the APP, one operation can determine whether or not a non-covered zero exists. A serial processor would have to search, as a minimum, all non-covered elements until a zero is found. If no non-covered zeros exist, then every non-covered element would have to be searched.

E) Next, it is determined whether or not a starred (*) zero exists in the same row as the primed (') zero, which is the case in Figure 3-4 E).

Since this is the case, the column for the starred zero is uncovered

and the row for the primed element is covered. These operations correspond to instructions 27 and 28 in Figure 3-5. The process is again repeated, beginning at instruction 23. However, this time, no non-covered zero's exist. Accordingly, as per Figure 3-5, the process jumps to instruction 39.

F) From the algorithm of Figure 3-5, the minimum of all non-covered elements is first determined. By referring to Figure 3-4 E), it can be seen that 2 is the minimum non-covered element. A square has been placed around the element 2 to make its position in the matrix clear for the reader. The numerical value for the minimum element is subtracted from all non-covered columns and added to all covered rows. Note that the values for the non-covered column elements that are covered by a row, do not change. These operations correspond to instructions 39 through 41 in Figure 3-5. These operations are most efficiently performed on the APP since the minimum of all non-covered elements, and the subtractions and additions can be performed globally, each within a single operation. To the contrary, a serial processor would have to perform each of these operations sequentially on all appropriate elements.

G) The operation returns to instruction 23. Again, there is a non-covered zero, shown with the square around it, which is primed.

H) Again, there is a starred zero in the row of the primed zero, so the cover (or line) in the column of the starred zero is removed and the row of the primed zero is covered, as is shown and as described by instructions 27 and 28.

I) The operation again returns to instruction 23 where a non-covered zero, indicated by the square, can be found. However, this time there is not a starred zero in the row of the primed zero. Accordingly, the next instruction jumps to 29.

J) First the column of the last primed element is searched to determine whether or not there is a starred zero in the column. In this case, there is a starred zero in the column, as is shown by the starred zero with the square around it. This starred zero will always have a primed element in its row.

These elements are tagged as "in sequence". These operations correspond to instructions 29, 30 and 31. Next, the operation is again started at instruction 29, using the last primed zero. However, this time the last primed element does not have a starred zero in its column, so the operation jumps to instruction 32. Again, the APP can determine the element with a starred or primed zero in the respective column and row. The serial processor would have to check each element in the column or row until the starred or primed zero is determined.

K) Each starred element in sequence is unstarred and its covered column is uncovered. Each primed element in sequence is starred. Again these operations can be performed globally with the APP. These operations correspond to instructions 33 and 34.

L) The columns for all starred elements are covered and a test is made to determine whether or not all columns are covered. In this example, all columns are covered and the optimization process is completed. The starred elements represent the assignments. If all columns had not been covered, all primes would be erased and all rows uncovered per instructions 37 and 38 and the process would be begun again at instruction 23. Again, all of these operations can be performed globally with the APP, but would have to be performed sequentially with the serial processor.

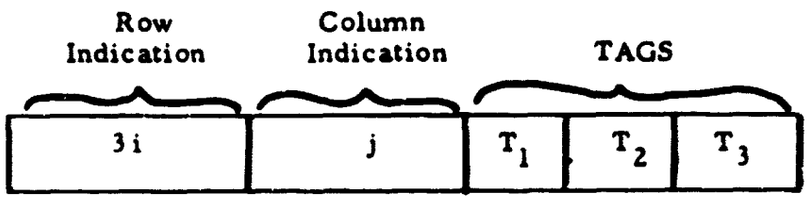
As is evident from this simple example, the APP is ideally suited for solving problems of this type. It can find maximum or minimum elements within any row or column, the entire matrix or any tagged subset of the matrix. It can

perform arithmetic operations simultaneously on these elements. It can chase through the matrix, going down appropriate rows and columns from tagged elements. These are the types of operations required to efficiently solve these types of problems.

3.3.2 Solution of the Binary Assignment Problem

3.3.2.1 APP Solution

In this section the binary assignment algorithm is implemented by associative processing techniques. The algorithm is described in Figure 3-6. Terminology is defined in Table 3-1. The format for data stored in the APP is shown below:



If it is impossible to assign some resource to some Task, i.e., if the matrix element value is zero, the element need not be stored in the memory. The flow diagram of Figure 3-7 illustrates the computational process.

A program, written in the associative machine language, described in subsection 3.2.2, is presented in Figure 3-8. Note that a total of 31 instructions are required. The number of instructions is invariant to the size of the rating matrix. It is only necessary to alter the limits of the A, B and D counters in the appropriate instructions as the size of the matrix varies.

The following three assumptions were made in working out the timing analysis.

1) There is an average of eight entries in each column of the rating matrix.

2) An average of five iterations through the loop provides a solution.

3) The time required for each instruction is as described in subsection 3.2.2.

There are two major timing loops in Figure 3-3 required for the row and column index iterations, namely:

1) Instructions Z6 $\longrightarrow$ Z12 for rows

2) Instructions Z12 $\longrightarrow$ Z19 for columns.

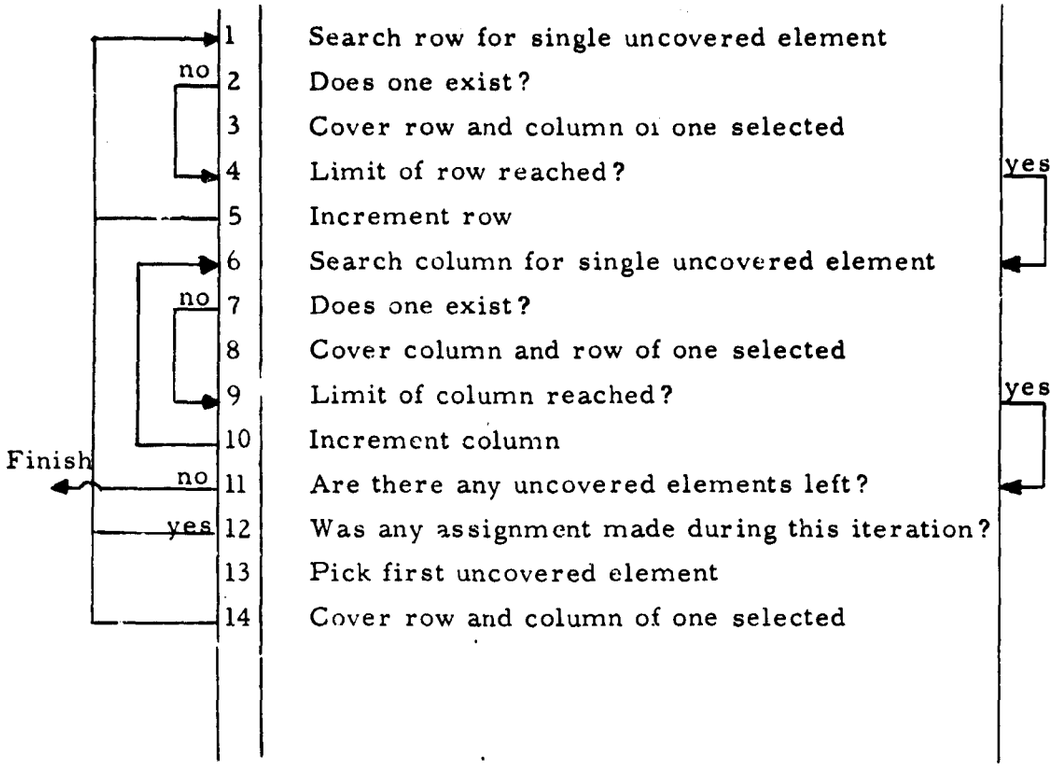


Figure 3-6 Fundamental Algorithm for Binary Assignment Problem

TABLE 3-1

TABLE OF TERMS FOR THE

BINARY ASSIGNMENT PROBLEM

i - $1 \leq i \leq n$, indicates a particular row in assignment matrix

j - $1 \leq j \leq n$, indicates a particular column in assignment matrix

p - an index on i

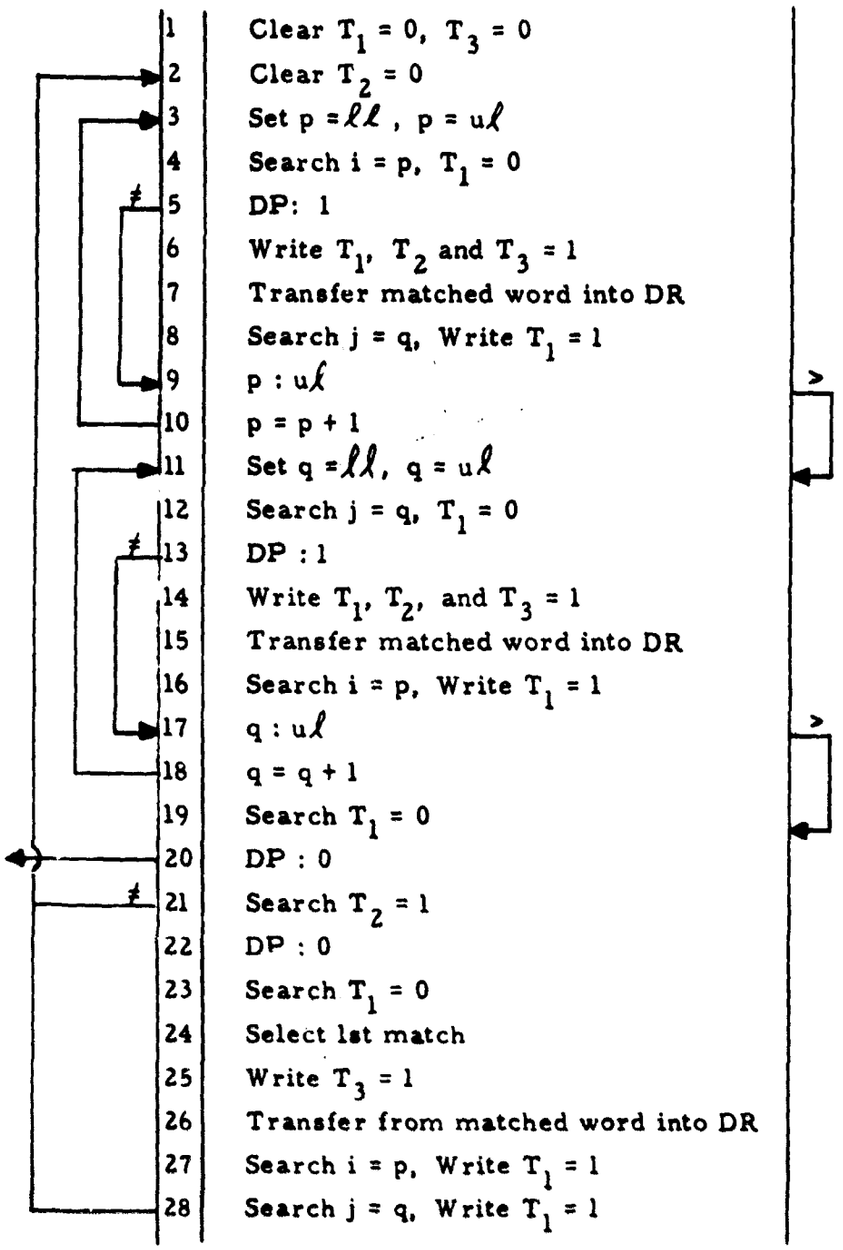
q - an index on j

n - indicates size of the $n \times n$ assignment matrix

T_1 - tags covered row or column

T_2 - indicates when at least one assignment is made during an iteration

T_3 - indicates an assignment and represents the optimum assignment



$T_1 = 1$, for a covered row or column

$T_2 = 1$, when at least one assignment is made during an iteration

$T_3 = 1$, when a null element is assigned and represents the optimum assignment

Figure 3-7 Flow Diagram of Program for the Binary Assignment Problem

Instruction Number	DP	A	B	T_1	T_2	T_3	D	Comments
Z-1	1	- - - -	- - - -	S 1 W	- - -	S 1 W	-	Clear T_1 and T_3 = 0
Z-2	Set D = i_{max}							Sets row and column index limit
Z-3	Set $A_{FL} = p_{max}$, $A_{IL} = p_{min}$							Sets digit line index for rows
Z-4	Set $B_{FL} = q_{max}$, $B_{IL} = q_{min}$							Sets digit line index for columns
Z-5	1	- - - -	- - - -	- - -	S 1 W	- - -	-	Clear T_2 = 0
Z-6	Read D into DR under A counter field control							
Z-7	1	S D W L	- - - -	S O W	- - -	- - -	I	Search i = p, T_1 = 0, increment D
Z-8	IF D ≠ 1, jump to Z-12							
Z-9	N	- - - -	- - - -	S O W	S O W	S O W	-	Write T_1, T_2 and T_3 = 1
Z-10	Transfer from matched word into DR							
Z-11	1	- - - -	S D W L	S O W	- - -	- - -	-	Search q for matched p, write T_1 = 1
Z-12	IF D ≠ 1, jump to Z-6							
Z-13	Transfer D into DR under B counter field control							
Z-14	1	- - - -	S D W L	- - -	S O W	- - -	I	Search j = q, T_1 = 0, increment D
Z-15	IF DP ≠ 1, jump to Z-19							
Z-16	N	- - - -	- - - -	S O W	S O W	S O W	-	Write T_1, T_2 and T_3 = 1, for match
Z-17	Transfer from matched word into DR							
Z-18	1	- - - -	S D W L	S O W	- - -	- - -	-	Search p for matched q, write T_1 = 1
Z-19	IF D ≠ 1, jump to Z-13							

Figure 3-8 Machine Language Program for Binary Assignment Problem

Instruction Number	DP	A	B	T_1	T_2	T_3	D	Comments
Z-20	1	- - - -	- - - -	S O W	- - -	- - -	-	Search $T_1 = 0$
Z-21	IF DP = 0, jump to S1							
Z-22	1	- - - -	- - - -	- - -	S O W	- - -	-	Search $T_2 = 0$
Z-23	IF DP ≠ 0, jump to Z-5							
Z-24	1	- - - -	- - - -	S O W	- - -	- - -	-	Search $T_1 = 0$
Z-25	Clear and select 1st match							
Z-26	N	- - - -	- - - -	- - -	- - -	S O W	-	Write $T_3 = 1$
Z-27	Transfer from matched word into DR							
Z-28	1	S D W L	- - - -	S O W	- - -	- - -	-	Search p for matched word, write $T_1 = 1$
Z-29	1	- - - -	S D W L	S O W	- - -	- - -	-	Search q for matched word, write $T_1 = 1$
Z-30	Jump to Z-5							
S-1	Stop							

Figure 3-8 (continued)

It is necessary to iterate through instructions Z-6, Z-7, Z-8 and Z-12, and Z-13, Z-14, Z-15 and Z-19, "n" times, multiplied by the number of iterations, but through either Z-9, Z-10, and Z-11, or Z-16, Z-17, and Z-18 a total of only n times independent of the number of iterations. Since these loops represent the principal timing requirement, the total time required can be expressed as

$$T = 2K\, n\, (1.75 + .050 \log_2 n) + n\, (2.0 + .050 \log_2 n)$$

where

n - is the number of columns or rows in the matrix

K - is the number of iterations required.

3.3.2.2 Comparison, APP with Serial Processor

As previously mentioned, a timing analysis for this problem, using two serial processors, the AN/FSQ-31(V) and the L-3500A, was available from a Librascope internal report. The data for each of these serial processors and the calculated data for the APP are presented in Table 3-2. To provide a reference, the L-3055A has a cycle time of 5.0 microseconds and requires 113 instructions for this problem and the AN/FSQ-31(V) has a cycle time of 2.5 microseconds and requires 84 instructions. The L-3055A and AN/FSQ-31(V) are the central processors used in the 473L and 465L command and control systems respectively.

The data of Table 3-2 are plotted in Figure 3-9 where the abscissa indicates the size of the matrix, the left hand ordinate indicates the solution time and the right hand ordinate indicates the solution time ratio of the APP over the two serial processors. For a 1000 × 1000 element matrix, the time for APP to solve the problem is less by a factor of 1000 than for either of the serial processors.

One criticism can be made in this comparison regarding the vintages of the serial processors versus the APP. More sophisticated serial processors are

ASSOCIATIVE PROCESSOR			AN/FSQ-31(V)		L-3055A	
n	Log_2 n	T	T	Ratio	T	Ratio
1000	10	25 ms	30 S	1200	80 S	3200
500	9	12.2 ms	2 S	160	6 S	490
100	7	2.12 ms	100 ms	47	300 ms	140
30	5	.66 ms	20 ms	30	50 ms	76
10	4	.21 ms	2 ms	-	5 ms	-

Table 3-2 Calculated Timing Data to Compare APP with AN/FSQ-31(V) and L-3055A Serial Processors for Binary Assignment Problem

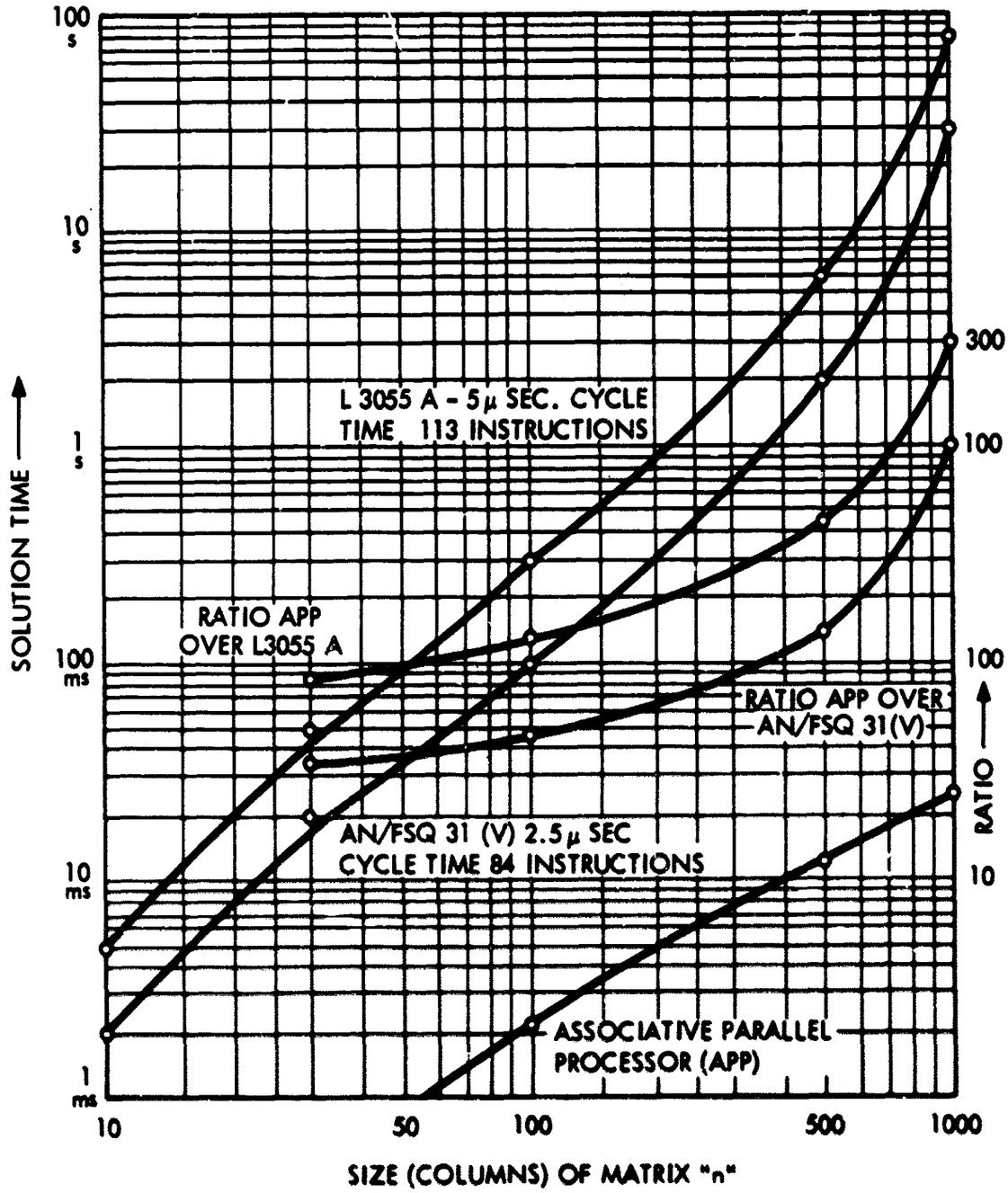


Figure 3-9 Comparison of APP with Two Serial Processors for Solution of Binary Assignment Problem

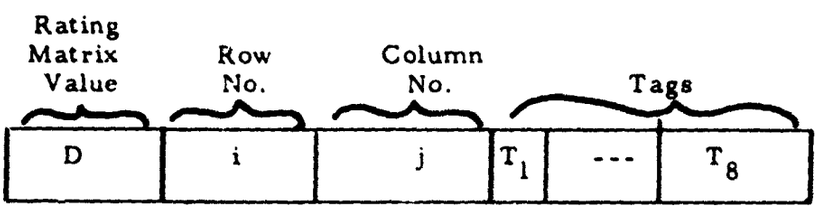
available which can produce processing speed improvements in the order of two to five over the AN/FSQ-31(V). However, even taking this factor into account, the APP would still have a significant speed improvement in solving the binary assignment problem.

3.3.3 Solution of the General Assignment Problem

3.3.3.1 APP Solution

The algorithm for the solution of the general assignment problem was presented in subsection 3.3.1. A more detailed program is developed in this subsection to determine the timing requirements.

The organization of the data within the associative memory of the APP is shown below.



Note that eight tag bits are required. When "D" has a value of zero in the weapon assignment problem, i.e., when a particular missile is not capable of reaching a particular target, it is not necessary to store the row and column indexes within the associative memory.

From Figure 3-5 in subsection 3.3.1, it can be seen that the algorithm for the general assignment problem is divided into five parts, namely:

1) Initial adjustment for maximizing or minimizing
2) Initial independent zero assignment
3) Cost matrix Test
4) Assignment adjustment
5) Cost matrix adjustment

Flow diagrams for each of these five parts of the algorithm were first developed. Terminology employed in the five flow diagrams is presented in Table 3-3. Note that tags not required in subsequent parts of the algorithm are re-used.

i - $i \leq i \leq n$, indicates a particular row in the assignment matrix

j - $1 \leq j \leq n$, indicates a particular column in the assignment matrix

p - an index on i

q - an index on j

h - indicates the size of the n × n assignment matrix

D - rating matrix value for a particular element

d - a binary digit index on D

d^i - the initial binary digit index on D

D^l - a limit search on D

D_d - a single digit search on D

Particular tags used in each of the five parts of the algorithm are:

1. Initial adjustment for minimizing
   - T_1 - tags a row (or column) for a particular iteration sequence
   - T_3 - tags used to indicate a digit borrow in subtraction operation
2. Initial independent zero assignment
   - T_1 - tags all zeros
   - T_2 - tags starred zeros
   - T_4 - tags covered columns
3. Cost Matrix Test
   - T_1 - tags all zeros
   - T_2 - tags starred zeros
   - T_4 - tags covered columns
   - T_5 - tags primed elements
   - T_6 - tags last primed element
   - T_7 - tags covered rows
4. Cost Matrix Adjustment
   - T_3 - tags smallest non-covered element in matrix
   - T_8 - used to indicate a digit borrow in the subtraction operation then a digit carry in the addition operation
5. Assignment Adjustment
   - T_2 - tags starred zeros
   - T_3 - tags starred zeros in sequence
   - T_4 - tags covered columns
   - T_5 - tags primed elements
   - T_6 - tags last primed element initially and then all primed elements in sequence
   - T_7 - Tags covered rows

At completion of algorithm, tag T_2 contains the elements representing the assignments.

Table 3-3 Definition of Terms for General Assignment Problem

Figure 3-10 presents the flow diagram for a minimizing assignment problem illustrated only for rows. The program for the columns would be exactly the same, except that column indexes would be specified. The flow diagrams for the initial independent zero assignment, the cost matrix test, the cost matrix adjustment and the assignment adjustment are shown in Figures 3-11, 3-12, and 3-13.

From the flow diagrams, the program in machine language was constructed and is presented in Appendix D. Note that 132 instructions are required. Note once again that the number of instructions is independent of the size of the cost matrix.

In working out the timing requirement, the following assumptions were made:

1. D can be represented by 10 significant binary digits
2. $\frac{\sqrt{N}}{N} \times 100\%$ of the N rows initially have no independent zeros
3. $\sqrt{N}$ iterations through cost matrix test loop (step 1)
4. $\frac{\sqrt{N}}{2}$ iterations through adjustment loop (step 3)
5. $\sqrt{N}$ iterations through assignment adjustment loop (step 2)

where D is the rating matrix value and N is the size of the N × N assignment matrix.

Using these assumptions, Table 3-4 was prepared to represent the basis for the timing calculations for the general assignment problem. Based on Table 3-4, the calculated timing values are shown in Table 3-5.

3.3.3.2 Comparison, APP with Serial Processor

The speed utility of the APP compared to a serial processor in solving the general assignment problem was next investigated. In order that the comparison could be made on machines representing the same technological vintage, a sophisticated serial processor representing the present state of the art was hypothesized. Assumptions made on this serial processor are:

1) The memory access time is 0.5 microsecond

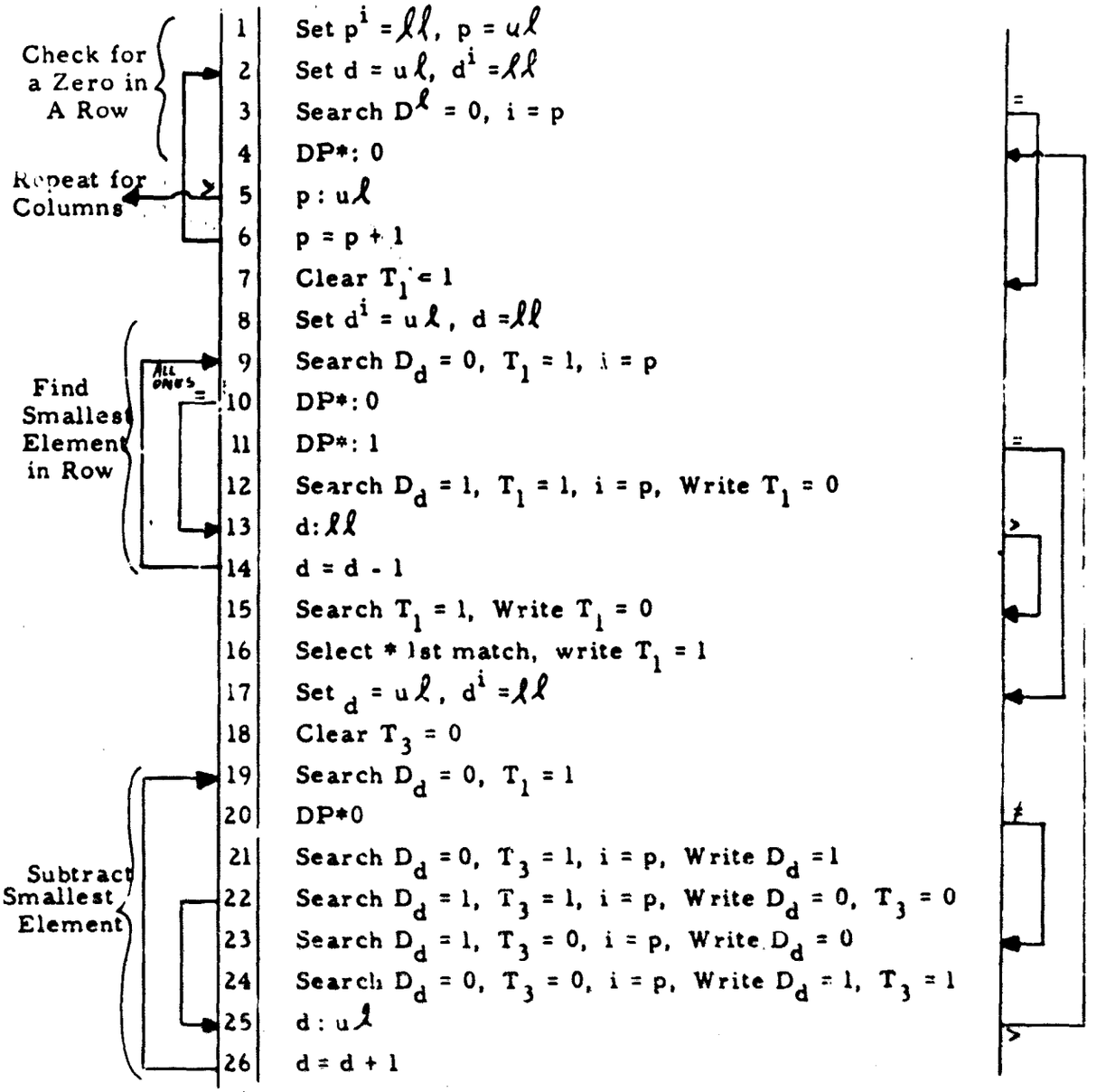


Figure 3-10 Flow Diagram for Initial Adjustment for Maximizing for rows, General Assignment Problem

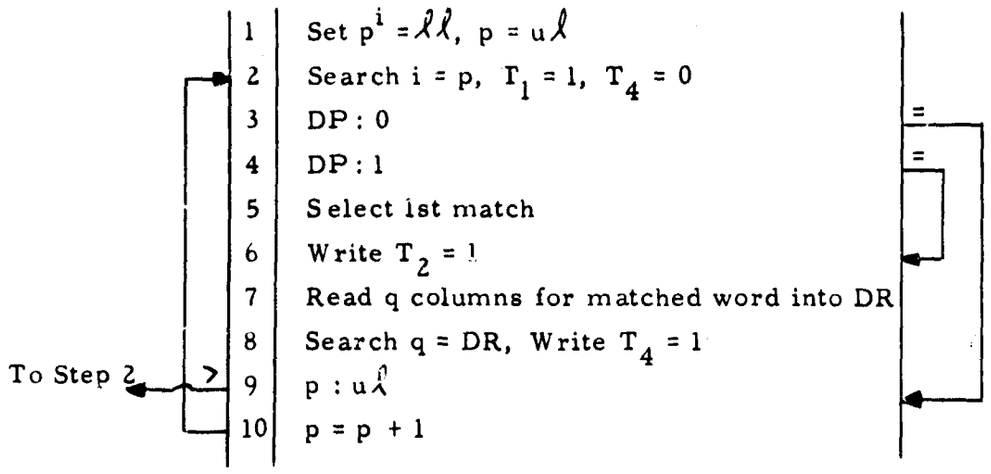


Figure 3-11 Flow Diagram for Initial Independent Zero Assignment, General Assignment Problem

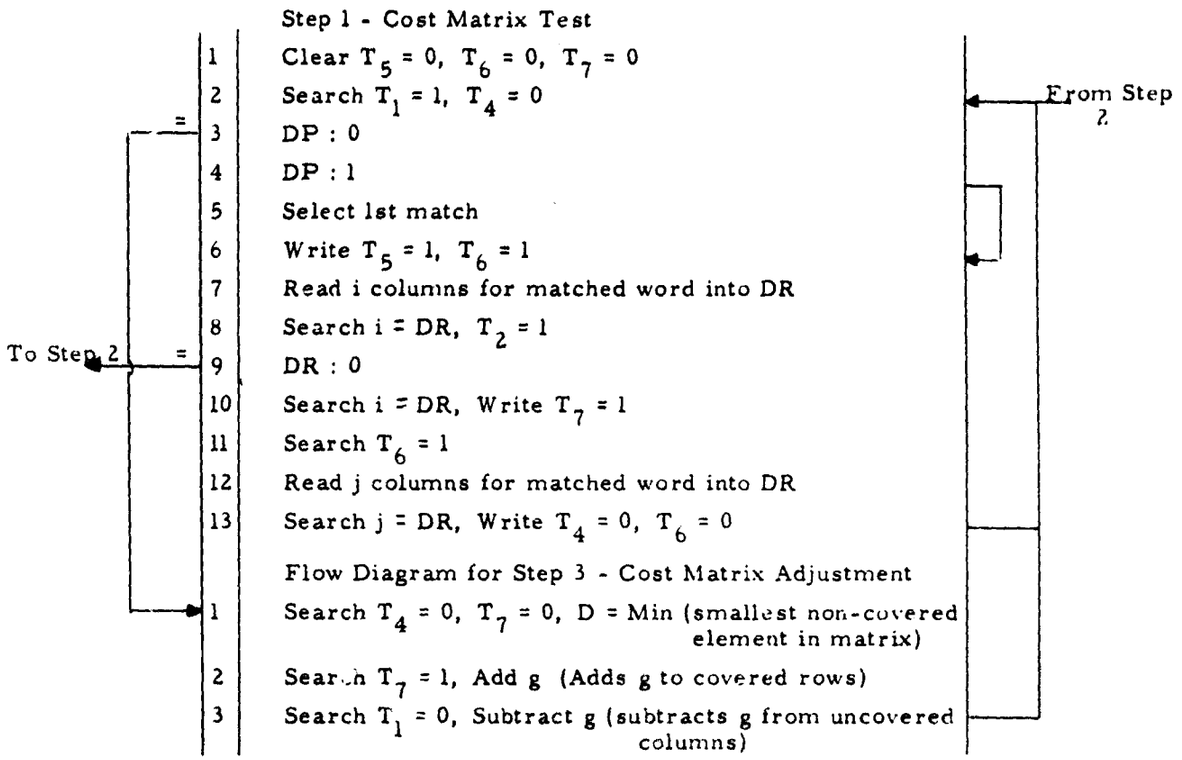


Figure 3-12 Flow Diagram for Step 1 - Cost Matrix Test and Step 3 - Cost Matrix Adjustment, General Assignment Problem

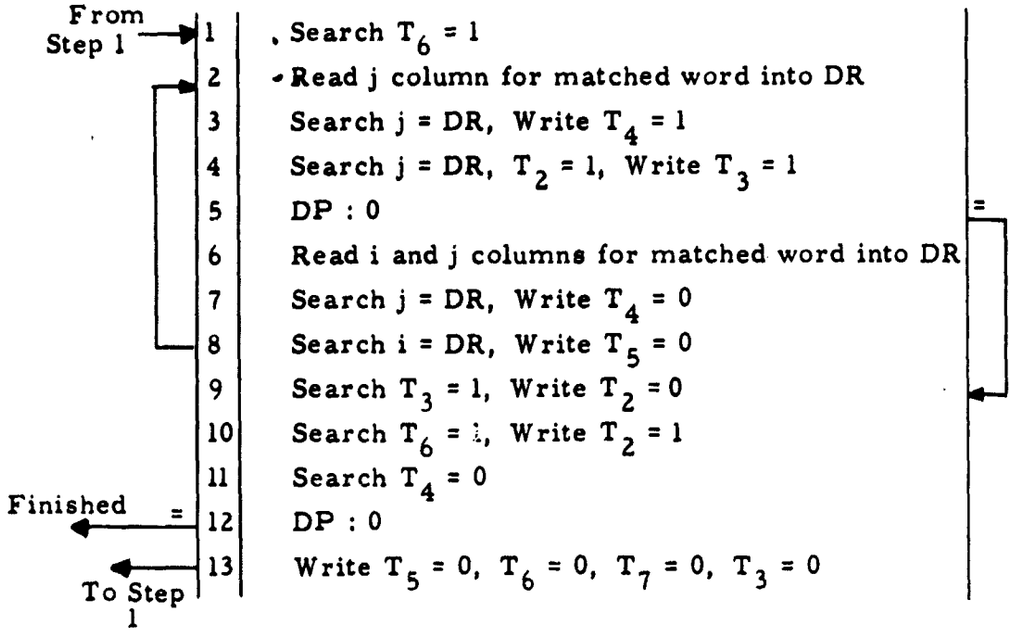


Figure 3-13 Flow Diagram for Step 2 - Assignment Adjustment, General Assignment Problem

Subroutine	Particular Section	Number of Iterations		Assumptions	Time per Iteration	Total Time
Initial Adjustment for Minimizing	Check for zero	2 N		Rows and columns	4A	SNA
	Smallest element in rows or columns	1.9N	.95N / .95N	5% of rows and columns have zero's; Find a min. halfway thru limit	3.5A $\log_2 D$	NA 3.28 $\log_2 D$
					2A $\log_2 D$	NA 1.9 $\log_2 D$
	Subtract	1.9 N		5% of rows and columns have zero's	A 3.5 $\log_2$	NA 6.56 $\log_2 D$
Initial Independent Zero Assignment		N	$N-\sqrt{N}$	$\frac{1-\sqrt{N}}{}$ has independent zero	6 1/2 A	$(N-\sqrt{N})$ (65A)
			$\sqrt{N}$	$\frac{\sqrt{N}}{N}$ has no independent zero's	3A	$\sqrt{N}$ 3A
Cost Matrix Test		$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$ iterations within loop and $\sqrt{N}$ changes in allocations	10A	N 10A
Cost Matrix Adjust	Minimum	$\frac{\sqrt{N}}{2}$	.5	Full D	3.5 A $\log_2 D$	3.0 $\frac{\sqrt{N}}{2}$ A $\log_2 D$
			.5	1/2 D	2.5 A $\log_2 D$	
	Add				3.5 A $\log_2 D$	$\frac{\sqrt{N}}{2}$ A 3.5 $\log_2 D$
	Sub				3.5 A $\log_2 D$	$\frac{\sqrt{N}}{2}$ A 3.5 $\log_2 D$
Assignment	Find sequence and tag	$\sqrt[2]{N}$	$\sqrt[3]{N}$		6A	$N^{5/6}$ A6
	Adjust	$\sqrt[2]{N}$			4.5 A	$\sqrt[2]{N}$ 4.5A

Table 3-4 Basis for Timing Calculation for General Assignment Problem for APP

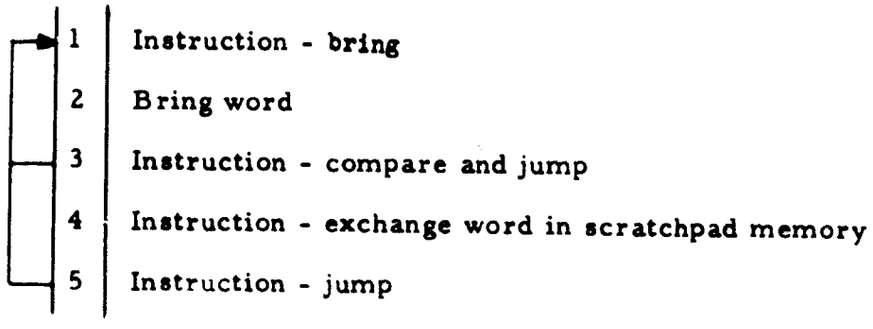
Subroutine	Formula \ N	30		50		100		200		500		1000	
Initial Adjustment for Minimizing	$(11.7 \log_2 D + 8)$ NA	N 1/2 3.5	1.87	N 1/2 7.1	3.12	N 1/2 10	6.25	N 1/2 14	12.5	N 1/2 22	31.2	N 1/2 32	62.5
Initial Independent Zero	$NA\left(6.5 - \frac{1}{\sqrt{N}} 3.5\right)$	N 5/6 17	.107	N 5/6 26	.175	N 5/6 46	.343	N 5/6 82	.675	N 5/6 177	1.66	N 5/6 318	3.3
Cost Matrix Test	N 10A		.150		.250		.500		1.0		2.50		5.00
Cost Matrix Adjustment	$\sqrt{N}$ A $5 \log_2 D$		.138		.178		.250		.350		.550		.880
Assignment Adjustment	$N^{5/6}$ A $6 + \sqrt[2]{N}$ A 4.5		.063		.094		.160		.278		.582		1.03
	TOTAL →		2.33 ms		3.8 ms		7.5 ms		14.8 ms		36.5 ms		72.7 ms

Table 3-5 Timing Calculation for General Assignment Problem for APP

2) The serial processor has scratchpad memory and instruction look-ahead so that the instructions and data can be accessed into the scratchpad memory during and in parallel with the execution of arithmetic and logical processing.

3) The arithmetic unit is the parallel type so that the processing operations required for this problem can be performed in less than the access time of 0.5 microsecond.

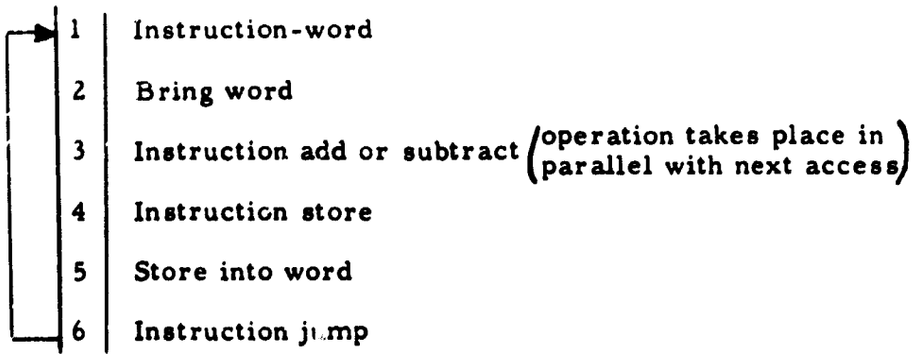
Two basic processing operations are required for the serial processing. The first involves accessing a word from memory and then making a comparison. An example of this operation is determining the minimum element in a row or column of the assignment matrix. The number of memory accesses required to perform this type of operation is shown below.



One memory access time is required to bring an instruction as well as to bring a word from memory. Assuming that the instruction interpretation and the arithmetic and logic operations are performed in parallel with the memory access, this instruction sequence is performed in five memory access times in the worst case, and three access times in the other case. It is assumed that four memory access times or two microseconds will be required, on the average, for this type of operation.

The second type of operation involves accessing a word from memory, subtracting or adding to the word contents and then storing the new value back into the word. An example of this operation is subtracting the minimum element in a row or column in the assignment matrix. The number of memory accesses

required to perform this type of operation is shown below:



In this case, six access times or three microseconds will be required.

Based on these assumptions and on the flow diagram of Figure 3-5, each of the five subroutines in the flow diagram will require the approximate times shown below, where

A_c - stands for access time of memory

N - represents the number of rows or columns in the rating matrix.

Initial adjustment for minimizing

TO COMPARE - $2N^2$ 4 A_c

TO SUBTRACT - $2N^2$ 6 A_c

Total Time $20 A_c N^2$

Initial independent zero assignment

Assumes $\frac{\sqrt{N}}{N}$ 100% of N rows initially have no independent zeros and, on the average, 1/2 N elements are required in rows or columns to find a non-covered zero.

TO FIND NON-COVERED ZERO - $1/2 N^2$ 4 A_c

TO COVER COLUMN OF STARRED ELEMENT - $N (N - \sqrt{N})$ 4 A_c

Total Time $6 A_c N^2 - 4 A_c N\sqrt{N}$

Cost Matrix Test

Assumes $\frac{\sqrt{N}}{2}$ iterations where no non-covered zeros exist, $\sqrt{N}$ iterations where non-covered zeros do exist, and $0.25 N^2$ elements must be searched on the average to find a non-covered zero when it exists.

WHEN NO NON-COVERED ZEROS EXIST - $N^2 \sqrt{\frac{N}{2}}\ 4\ A_c$

WHEN NON-COVERED ZEROS DO EXIST - $.25\ N^2\sqrt{N}\ 4\ A_c$

Total Time $3\ A_c N^2\sqrt{N}$

Cost Matrix Adjustment

Assumes $\sqrt{\frac{N}{2}}$ iterations

TO FIND SMALLEST ELEMENT IN MATRIX $N^2\sqrt{\frac{N}{2}}\ 4\ A_c$

TO SUBTRACT OR ADD SMALLEST ELEMENT $N^2\sqrt{\frac{N}{2}}\ 6\ A_c$

Total time $5\ A_c N^2 \sqrt{N}$

Assignment Adjustment

Assumes $\sqrt{N}$ iterations and $\sqrt[4]{N}$ linkages of starred elements in sequence

No linkages - $\sqrt[4]{N}$

No. of column covers removed per linkage - N

Total Time $(\sqrt[4]{N}\ \ N\sqrt{N})\ 4\ A_c$

The timing requirements for each of the five subroutines are tabulated in Table 3-6 for several cost matrix sizes. The next to last row indicates the total time required in milliseconds for the matrix sizes shown, and the last row indicates the speed improvement of the APP over this sophisticated serial processor for matrix sizes shown. As shown, for a 1000 × 1000 matrix, the serial processor would require 139 seconds or 2.3 minutes to solve the general assignment problem, whereas the APP would require 72.7 milliseconds, representing a speed improvement of 1910 times.

3.3.4 Solution of the Transportation Problem

3.3.4.1 APP Solution

The algorithm for the solution of the transportation problem is presented in Figure 3-14. Note that the algorithm is again divided into five parts, where adjustment for minimizing or maximizing and the cost matrix adjustment parts

Times Shown in Milliseconds

Subroutine	Formula N	30	50	100	200	500	1000
Initial Adjustment for Minimizing	$10 N^2$	9.0	25.0	100	400	2500	10,000
Initial Independent Zero Assignment	$3 N^2 - 2N$ N	3.0	8.2	32	126	772	3,000
Cost Matrix Test	$1.5 N^2$ N	7.35	27.0	150	848	1,950	47,200
Cost Matrix Adjustment	$2.5 N^2$ N	12.3	45	250	1410	13,500	78,700
Assignment Adjustment	$2 N^{7/4}$	.8	1.9	6	23	105	354
	TOTAL	32.5	107.0	538.0	2810	24,800	139,000
	Ratio Compared To Add	14	28	72	190	680	1910

Table 3-6 Timing Calculation for General Assignment Problem for an Assumed Sophisticated Serial Processor

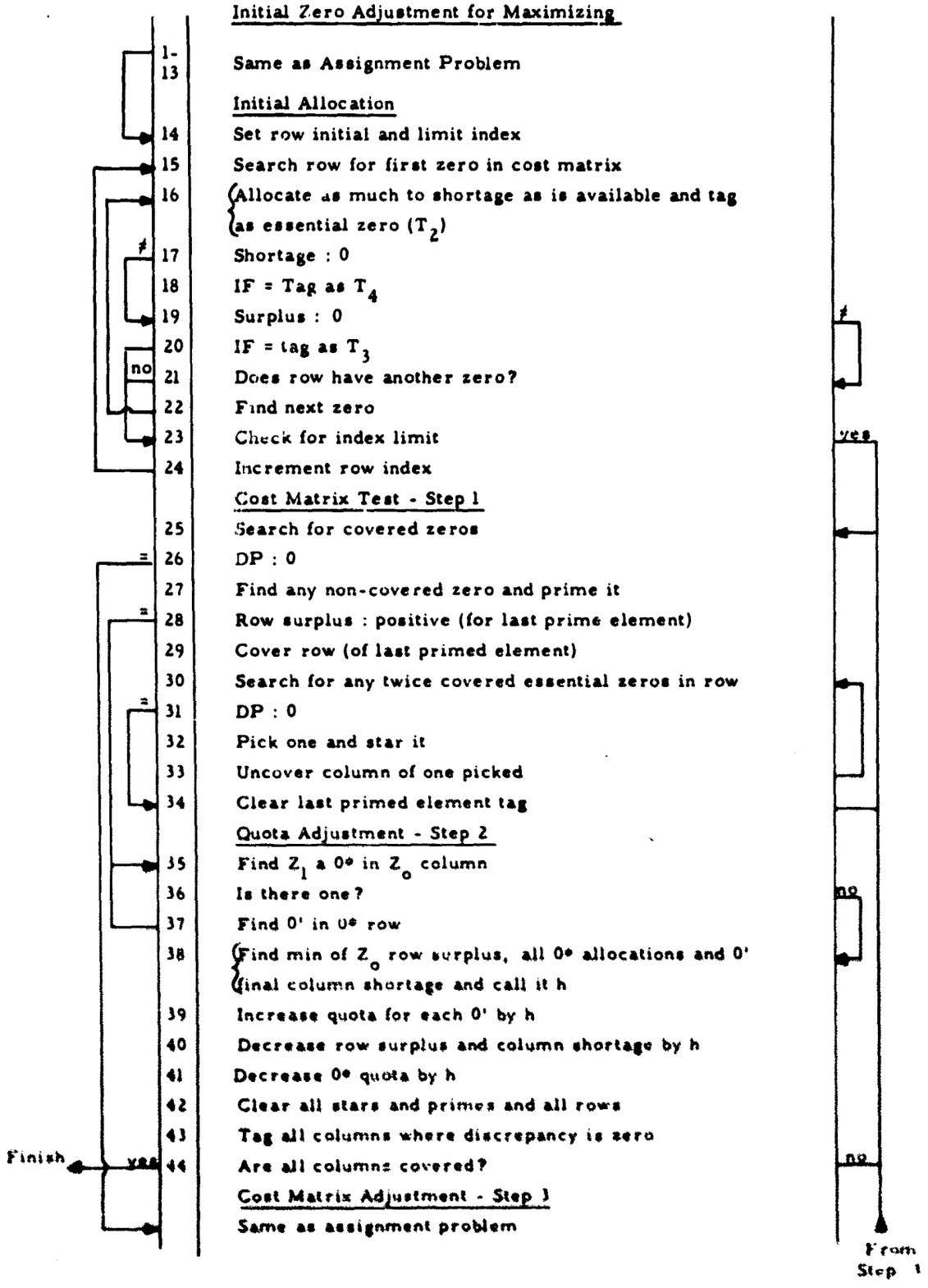


Figure 3-14 Transportation Problem Algorithm

are identical to these same parts in the assignment problem. Terms used in Figure 3-14 not previously defined are defined below:

Quota - indicates the amount of commodity assigned to a particular element in the assignment matrix.

Surplus - indicates the amount of commodity remaining at a particular source. At the beginning of the algorithm, the summation of all surpluses represents the total of the commodities available. At the conclusion of the algorithm, all surplus values are zero, since all surplus commodities at sources will have been assigned to the destinations.

Shortage - indicates the amount of commodity yet required for a particular destination. At the beginning of the algorithm, the summation of all shortages represents the total of the commodities available.

At the conclusion of the algorithm, all shortage values are zero, since all available commodities at the sources will have been assigned to the destinations.

Essential Zero - indicates a zero in the cost matrix whose quota is positive (non-zero).

Also, as in the preceding algorithm, certain rows or columns of the cost matrix are distinguished by covering them, certain zero elements in the cost matrix are distinguished by starring (*) them, and others are distinguished by priming (') them. Definition of tags and other terms used in Figure 3-14 are:

T_1 - a covered column

T_2 - tags essential zeros

T_3 - tags rows where surplus = 0

T_4 - tags column where shortage = 0

Z_o - denotes an uncovered 0' (there will be one at point where listed in algorithm)

Z_1 - denotes a 0* in Z_o's column

A statement of the algorithm in words which helps to clarify Figure 3-14 is presented below. The statement begins with the initial allocation point in the algorithm.

Initial Allocation - Find a zero element Z in the cost matrix. If both its surpluses and shortages are positive, allocate a quota to Z in the amount of the smaller of the two, and reduce each by that same amount. Cover a column when its shortage becomes zero. Repeat, for each zero in the cost matrix.

Cost Matrix Test - Step 1 - Choose a non-covered zero and prime it. Consider the row containing it. If the surplus for this row is positive go at once to Step 2. If the surplus for this row is zero, cover the row, then star each twice-covered essential zero Z in the row and uncover Z's column. Repeat, unless all zero's are covered, at which time go to Step 3.

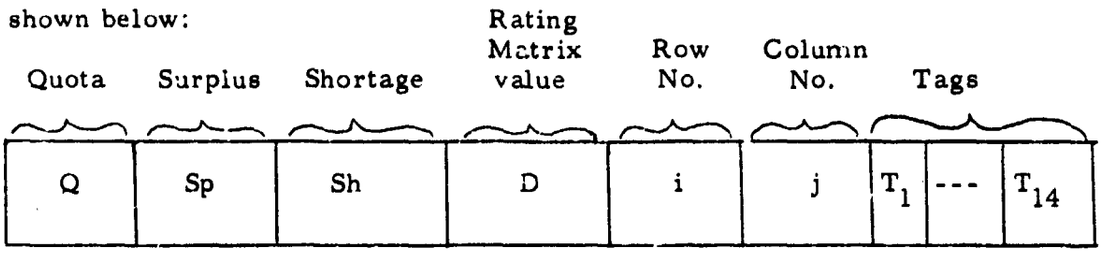
Quota Adjustment - Step 2 - Construct a sequence of alterna ing starred and primed zeros as follows: Let Z_o denote the uncovered 0' (there is only one). Let Z_1 denote the 0* in Z_o's column (if any). Let Z_2 denote the 0' in A_1's row. Let Z_3 denote the 0* in Z_2's column (if any). Continue until the sequence stops at a 0', Z_{ij}, which has no 0* in its column. (This sequence is unique since no column contains more than one 0*, and no row more than one 0').

The surplus in Z_o's row is non-zero, the shortage in Z_{ij}'s column is non-zero and the quota assigned to each 0* in the sequence $Z_o \ldots Z_{ij}$ is non-zero. Let h be the smallest of these positive numbers. Increase the quota of each 0' in the sequence by h, and decrease the quota of each 0* in the sequence by h. Erase all stars and primes, uncover all rows, and cover every column whose shortage is zero. Return to Step 1.

Cost Matrix Adjustment - Step 3 - Find g, the smallest non-covered element in the cost matrix. Add g to every covered row and subtract g from every uncovered column. Return to Step 1, not altering any stars, primes or

covered lines.

The organization of the data within the associative memory of the APP is shown below:



In comparison to the general assignment problem, note that Q, Sp and Sh additional data columns are required as well as six additional tag bits. Also, as with the general assignment problem, it is not necessary to store data for elements where the rating matrix value "D" is zero. These types of elements would correspond to missiles that could not reach a particular target in the weapon assignment problem.

Definitions for tags used for each of the five parts of the algorithm are described below.

1 Initial adjustment for minimizing

T_1 - tags a row (or column) for a particular iteration sequence

T_3 - tag used to indicate a digit borrow in subtraction operation

2. Initial allocation

T_1 - tags a covered column

T_2 - tags essential zero elements in cost matrix

T_3 - tags rows where surplus = 0

T_4 - tags columns where shortage = 0

T_5 - tags all cost matrix elements with zero value

T_6 - temporary shortage for last essential zero

T_7 - tags covered rows

T_8 - used to indicate that row has only one remaining zero

3. Cost Matrix Test (Step 1)

T_1 - tags a covered column

T_3 - tags rows where surplus is zero

T_4 - tags columns where shortage is zero

T_5 - tags all cost matrix zero elements

T_6 - tags last primed element

T_7 - tags covered rows

T_9 - tags all primed elements excluding the last one from Step 2

4. Cost Matrix Adjustment (Step 3)

T_1 - tags a covered column

T_5 - tags all cost matrix elements with zero value

T_7 - tags a covered row

T_8 - used as digit carry or borrow in addition or subtraction operation

5. Quota Adjustment (Step 2)

T_1 - tags a covered column

T_2 - tags essential zeros

T_3 - tags rows where surplus = 0

T_4 - tags columns where shortage = 0

T_6 - temporary storage for last essential zero

T_7 - tags covered rows

T_{10} - tags a star (*) on a line segment

T_{11} - indicates the last tagged 0*

T_{12} - tags a prime (') on a line segment

T_{13} - indicates the last tagged 0'

T_{14} - tags the smallest element in the sequence

Figure 3-10, the flow diagram for the "initial adjustment for minimizing" part of the algorithm for the assignment problem is applicable to the transportation problem. Figure 3-15 indicates the flow diagram for the initial allocation part of the transportation problem algorithm, Figure 3-16, the cost matrix test and cost matrix adjustment parts and Figure 3-17, the quota adjustment part.

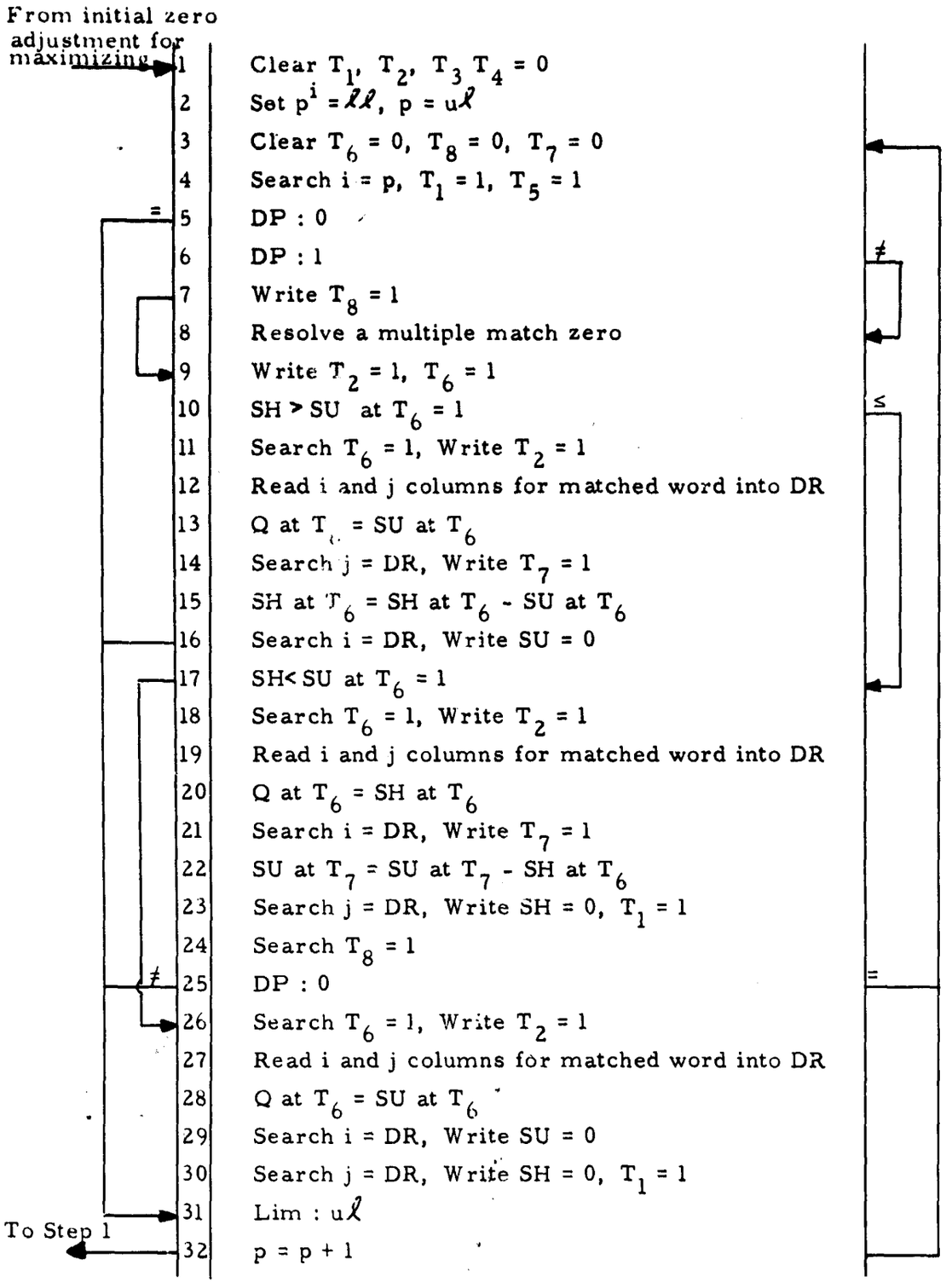


Figure 3-15 Flow Diagram, Initial allocation, Transportation Problem

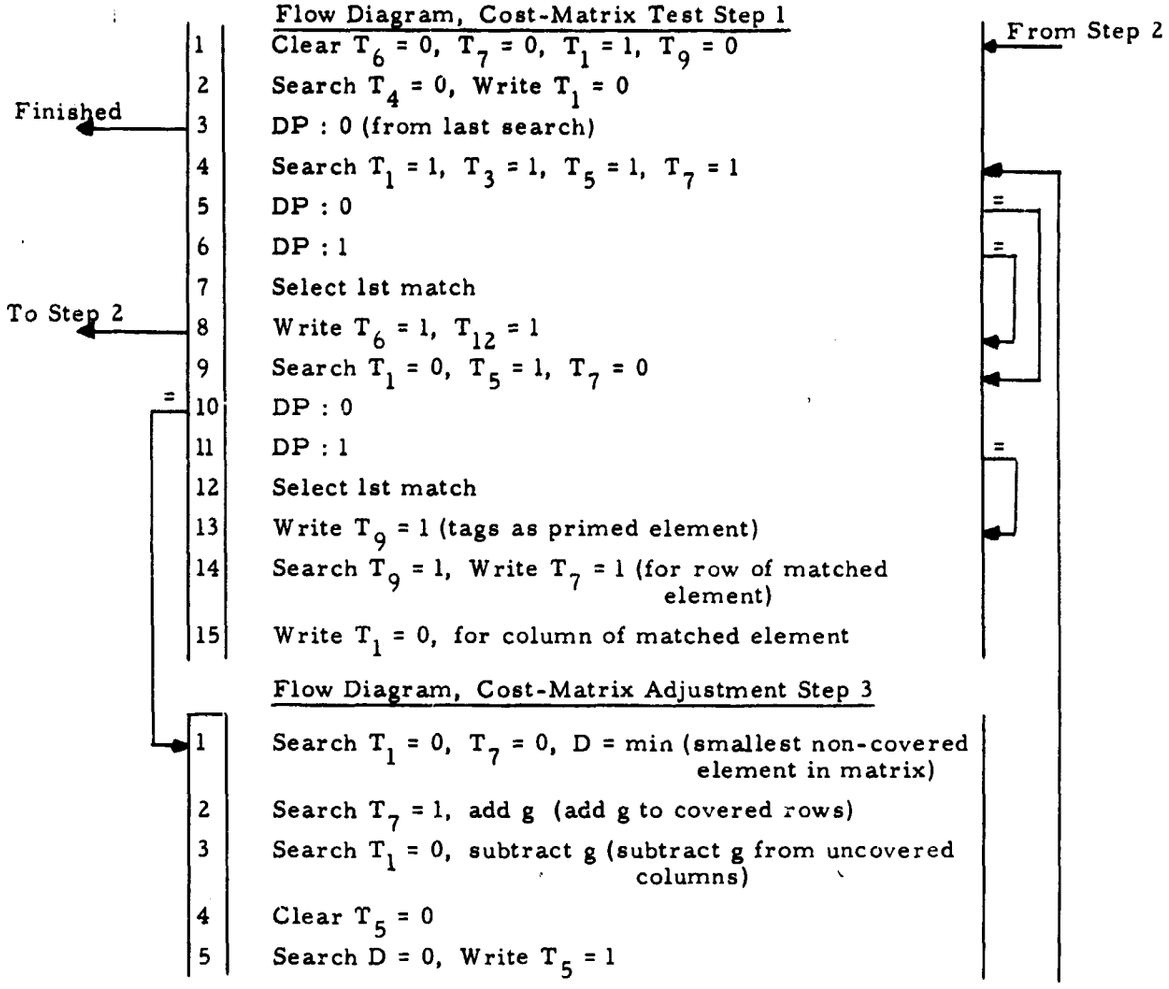


Figure 3-16 Flow Diagram, Cost-Matrix Test - Step 1 and Cost-Matrix Adjustment - Step 3 Transportation Problem

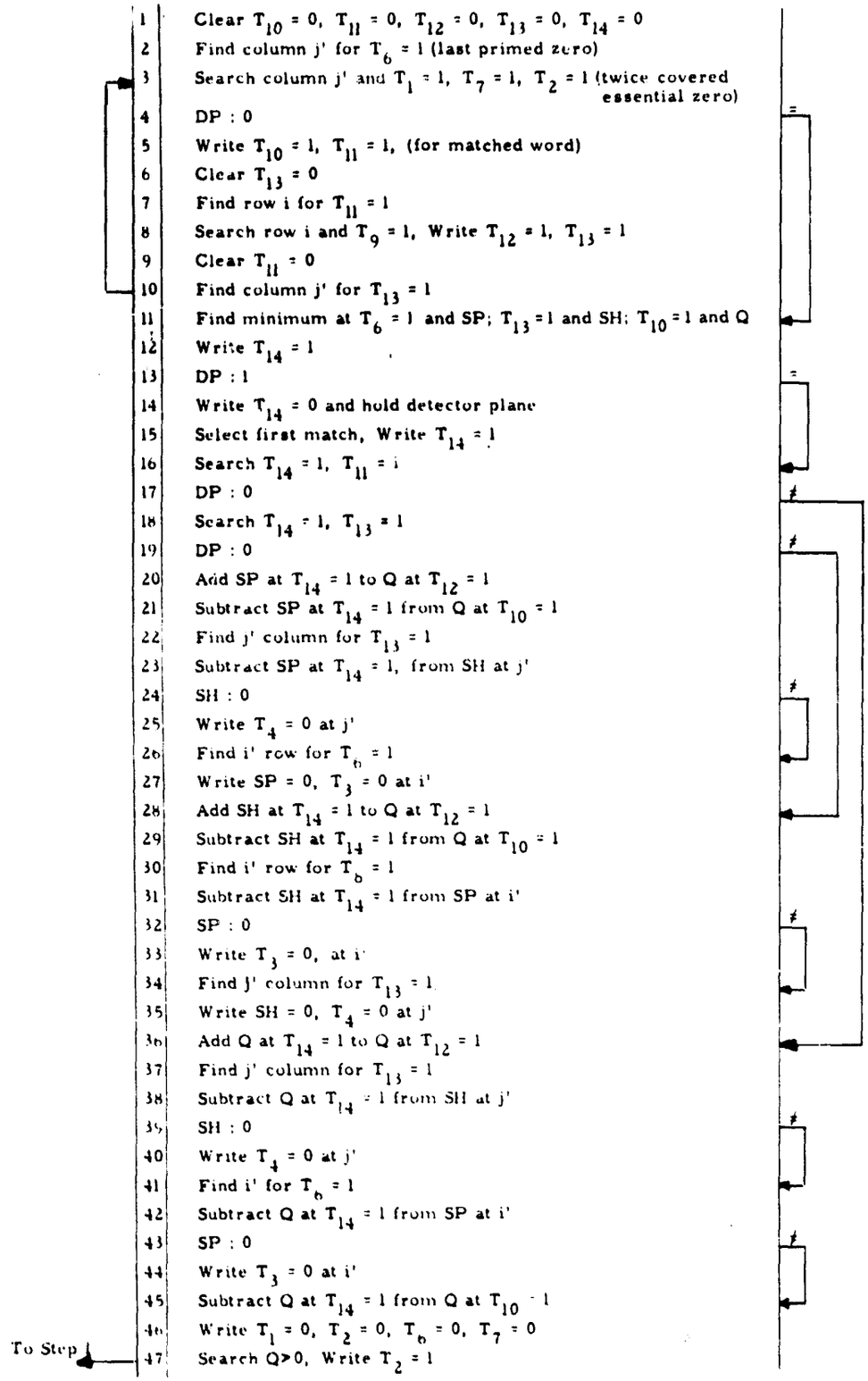


Figure 3-17 Flow Diagram, Quota Adjustment - Step 2
Transportation Problem

More machine language instructions would be required for the transportation problem than for the assignment problem. However, all of the fundamental processing operations have been defined in the machine language program for the assignment problem. Accordingly, a machine language program was not written for the transportation problem.

The timing requirement for the transportation problem was worked out in terms of the flow diagrams.

The following assumptions were made:

1) D, the rating matrix value, can be represented by 10 significant binary digits.
2) For convenience, the rating matrix will be assumed to be an $N \times N$ square. Note that the square matrix is not a requirement for the transportation problem as it was for the assignment problem.
3) Initially, the average number of surpluses per source for each row or of shortages per destination for each column is $\sqrt{N}$. Note that with this assumption an analogous assignment problem would require a cost matrix of size $N\sqrt{N} \times N\sqrt{N}$.
4) In the initial allocation loop, there are an average of $\sqrt[4]{N}$ allocations made per row.
5) For Step 1 - cost matrix test loop, there is an average of $\sqrt{N}$ consecutive iterations within the loop, but repeated $\sqrt{N}$ times.
6) The Step 3 - cost matrix adjustment loop is iterated $\sqrt{\frac{N}{2}}$ times.
7) The Step 2 quota adjust loop is iterated $\sqrt{N}$ times.

Using these assumptions, Table 3-7 was prepared, representing the basis for the timing calculations for the transportation problem. Based on Table 3-7, the calculated timing values are shown in Table 3-8.

3.3.4.2 Comparison APP with Serial Processor

The speed utility of the APP compared to a serial processor in solving the transportation problem was next investigated. The same serial processor,

Sub Routine	Section Within Loop	Number of Iterations		Assumptions	Time per Iteration	Total Time
Initial Zero Adjustment	Same as in Table 3-4					
Initial Allocation		$\sqrt{N}$	$\sqrt[4]{N}$	Average of $\sqrt{N}$ allocations per column but only $\sqrt[4]{N}$ can be made	$\sqrt[3]{N}(5.5+3\log_2 N)$ $+13.5+11.5\log_2 N$	$\left[\sqrt[3]{N}(5.5+3\log_2 N)\right.$ $\left.+13.5+11.5\log_2 N\right]$ $A\sqrt{N}$
Cost Matrix Test		$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$ per row number of changes in allocations and $\sqrt{N}$ iterations within loop	$(7.5+2\log_2 N)$	$NA(7.5+2\log_2 N)$
Cost Matrix Adjustment	Minimum	$\frac{\sqrt{N}}{2}$	.5	Full $\bar{D}$	$3.5\,A\log_2\bar{D}$	$3.0\frac{\sqrt{N}}{2}A\log_2\bar{D}$
			.5	$1/2\,\bar{D}$	$2.5\,A\log_2\bar{D}$	
	Add	$\frac{\sqrt{N}}{2}$		Half the time this loop is required to adjust rating matrix	$3.5\,A\log_2\bar{D}$	
	Sub	$\frac{\sqrt{N}}{2}$		"	$3.5\,A\log_2\bar{D}$	
	New zero's	$\frac{\sqrt{N}}{2}$			$\log_2 Q+1$	
					TOTAL	$\frac{A\sqrt{N}}{2}(10\log_2 D$ $+\log_2 Q+1)$
Quota Adjustment	Find trace and tag	$\sqrt{N}$	$\sqrt[3]{N}$		$5.5+2\log_2 N$ $+\log_2 Q$	
	Adjust	$\sqrt{N}$			$13.5+11.5$ $\cdot\log_2 Q$	
					TOTAL	$A\sqrt{N}\left[\log_2 Q+13.5\right.$ $\left.+\sqrt[3]{N}(5.5+2\log_2 N)\right]$

Table 3-7 Basis of Timing Calculation for Transportation Problem with APP

Times Shown in Milliseconds

Subroutine	Formula (N →)	30	50	100	200	500	1000
Initial Zero Adjustment	$(11.7 \log_2 D + 8)\, NA$	1.87	3.12	6.25	12.5	31.2	62.5
Initial Allocation	$N \sqrt[4]{N}\, A\, (12 + 3 \log_2 Q + 3.5 \log_2 D)$	1.8	3.7	9.0	37.4	70.5	181
Rating Matrix Test	$AN\, (7.5 + 2 \log_2 N)$	.18	.36	.83	1.85	5.1	11.3
Quota Adjustment	$\frac{A\sqrt{N}}{2}\, (10 \log_2 D + \log_2 Q + 1)$	.15	.21	.28	.4	.8	1.0
Rating Matrix Adjustment	$A\sqrt{N}\, [\sqrt[3]{N}\, (5.5 + 3 \log_2 N) + 13.5 + 11.5 \log_2 N]$	.37	.60	1.08	1.95	4.2	7.7
D = 1025, Q = 7	TOTAL	4.4	8.1	19.5	54	112	264

Table 3-8 Timing Calculation for Transportation Problem with APP

hypothesized for the general assignment comparison described in the last subsection, was assumed for this comparison. The transportation problem also requires the same two basic processing operations for serial processing as there described. Therefore, the same basic timing cycles are assumed as there described.

Based on these assumptions and on the flow diagram of Figure 3-14, each of the five subroutines in the macro flow diagram will require the approximate times shown below, where

A_c - Stands for access time of memory and is assumed to be 0.5 microsecond

N - represents the number of rows or columns in the rating matrix

Initial adjustment for minimizing (same as assignment problem)

To Compare	$2N^2\ 4\ A_c$
To Substract	$2N^2\ 6\ A_c$
TOTAL	$20\ A_c\ N^2$

Initial allocations

Assumes an average of $\sqrt[4]{N}$ allocations made per row and an average of 3/4 N elements down a row before either shortage or surplus becomes zero.

To search for zeros - $3/4\ N^2\ 4\ A_c$

Allocate to Shortage

Greater than compare	$-\ N\sqrt[4]{N}\ 4\ A_c$
Substract	$-\ N\sqrt[4]{N}\ 6\ A_c$
Update surpluses and Shortages	$-\ 2N^2\sqrt[4]{N}\ 4\ A_c$
TOTAL	$4\ A_c N^2 + 18 A_c\ N\sqrt[4]{N} + 8\ A_c N^2\ \sqrt[4]{N}$

Cost Matrix Test - Step 1

Assumes $\frac{N}{2}$ iterations where surplus for row of non-covered zero is

positive, $\sqrt{N}$ iterations where all zeros are covered and on the average $0.25\,N^2$ elements are searched to find a non-covered zero with a positive surplus in its row.

When all zeros are covered -

Finding Zeros - $N^2\sqrt{N}\,4\,A_c$

Row Surplus Test - $N^2\sqrt{N}\,6\,A_c$

Neglect remaining operations - 0

Non-covered Zero with positive row surplus

Finding zeros - $0.25\,N^2\frac{\sqrt{N}}{2}\,4\,A_c$

Row surplus positive test - $0.25N^2\,\frac{\sqrt{N}}{2}\,6\,A_c$

TOTAL $11.25\,A_cN^2\sqrt{N}$

Cost Matrix Adjustment - Step 3

Assumes $\frac{\sqrt{N}}{2}$ iterations

Search for minimum non-covered element - $4\,A_cN^2\frac{\sqrt{N}}{2}$

Subtract or add smallest element - $6\,A_cN^2\frac{\sqrt{N}}{2}$

TOTAL $5\,A_cN^2\sqrt{N}$

Quota Adjustment - Step 2

Assumes $\sqrt{N}$ iterations and $\sqrt[3]{N}$ sequences on the average

To determine sequence - $\sqrt[3]{N}\sqrt{N}\;N\,4\,A_c$

Final minimum element in sequence - $\sqrt[3]{N}\sqrt{N}\;6\,A_c$

Arithmetic operations on sequence - $\sqrt[3]{N}\sqrt{N}\;6\,A_c$

Remove row and column tags and primed and starred elements - $N^2\sqrt{N}\,4\,A_c$

TOTAL $12\,A_cN^{4/3} + 4\,A_cN^{11/16} + 4\,A_cN^{5/2}$

The timing requirements for each of these five subroutines are tabulated in Table 3-9 for several cost matrix sizes. The next to last row indicates the total time required in milliseconds for the matrix sizes shown, and the last

Time Shown in Milliseconds

Subroutine	Formula \ N	30	50	100	200	500	1000
Initial Adjustment for Minimizing	$10\,N^2$	9.0	25.0	100	400	2500	10,000
Initial Allocation	$(1.5 + 4\sqrt[4]{N})\,N^2 + 9\,N^{5/4}$	9.7	32.0	141	659	5080	24,000
Cost Matrix Test	$5.6\,N^{5/2}$	27.4	102	560	3160	29,600	176,000
Cost Matrix Adjustment	$2.5\,N^{5/2}$	12.3	45	250	1410	13,300	78,700
Quota Adjustment	$2N^{5/2} + 2N^{11/16} + 6N^{4/3}$	10.8	38.6	210	1160	10,800	63,600
	TOTAL	69	242	1261	6790	60,500	352,000
	Ratio Compared to APP	16	30	65	125	530	1330

Table 3-9 Timing Calculation for Assignment Problem for an Assumed Sophisticated Serial Processor

row indicates the speed improvement of the APP over the serial processor for matrix sizes shown. As shown, for a 1000 × 1000 matrix where it has been assumed that these are $N \times \sqrt{N}$ or 31,600 total allocations made, the sophisticated serial processor would require 352 seconds or 5.9 minutes to solve the transportation problem, whereas the APP would require 264 milliseconds representing a speed improvement of 1330 times.

3.3.5 Timing Comparisons for Network Flow Problems

Table 3-10 is a summary of the timing requirement for each of the three variations of the network flow problem, and illustrates the timing efficiency of the APP over a serial processor. However, the data shown for the serial processor should be interpreted as being only representative. Actual values would depend upon the particular serial processor employed and on the degree of cleverness used in constructing the program. In addition, the reader should bear in mind that the solution times shown are based on assumptions made on the number of iterations required within various parts of the algorithms. Even though the size of the cost matrix might remain constant, particular solution times would vary as the initial values of the variables change. However, the assumptions were made on the basis of representing a typical case for a problem statement. Nevertheless, the same assumptions were maintained in working out the timing for both the APP and serial processor. Accordingly, even though the solution times would vary from problem to problem, the ratios of efficiency shown should remain relatively constant.

From Table 3-10, note that the timing requirement in the three problem cases for the APP is increasing approximately linearly, or in direct proportion to "N", the order of the cost matrix, where the cost matrix was assumed square for the transportation problem. In contrast, the timing requirement in the three problem cases for the serial processors is increasing nearly in proportion to N^3, or the cube of the size of the matrix. This, then, represents the efficiency factor of the APP over a serial processor for network flow problems, of which the

SIZE N OF N-N MATRIX	BINARY ASSIGNMENT PROBLEM			GENERAL ASSIGNMENT PROBLEM			TRANSPORTATION PROBLEM		
	APP	Serial Process AN/FSQ31V	Ratio	APP	Serial Process	Ratio	APP	Serial Process	Ratio
30	66ms	20ms	30	2.3ms	32.5ms	14	4.4ms	69ms	16
50	-	-	-	3.8ms	107ms	28	8.1ms	242ms	30
100	2.1ms	100ms	47	7.5ms	538ms	72	19.5ms	1.26s	65
200	-	-	-	14.8ms	2.8s	190	54ms	6.8s	125
500	12.2ms	25	160	36.5ms	24.8s	680	112ms	60.5s	530
1000	25ms	30s	1200	72.7ms	139s	1910	264ms	352s	1350

Table 3-10 Summary of Timing Data for Network Flow Problems

weapon assignment problem is an example.

However, of even greater importance for the weapon assignment problem, is the fact that the APP can determine optimum solutions in a matter of milliseconds, which must be considered as virtually real time for the weapon assignment problem.

In contrast, where the matrix is large, the solution time for a serial processor would require in the order of seconds for the binary assignment problem, and in the order of minutes for the transportation problem. Obviously, for realistic weapon assignment problems which do require large matrices, a serial processor is not capable of providing real time solutions. In a time of national emergency, where pertinent initial data in the matrix could change right up to the instant of the weapons assignment and release thereof, the real time solution capability is essential. This capability is available with the APP and is not available with a serial processor.

3.4 SUMMARY AND CONCLUSIONS

The applicability of an APP for the solution of network flow problems and, in particular, the weapon assignment problem, was investigated. The network flow problem was couched within the model of the so-called Hitchcock-Koopmans transportation problem. Three variations on this model were considered as is listed below in the order of model complexity.

1. Binary Assignment Problem - This is a simplified version of the assignment problem, where the cost matrix has a value of only zero or one. The cost matrix is square.

2. Assignment Problem - This is a simplified version of the transportation problem, where each source initially has available only one unit of the commodity, and each destination has a requirement of only one unit of the commodity. The cost matrix is square.

3. Transportation Problem - In this case, the source initially has available

one or more units of the commodity, and each destination has a requirement of one or more units of the commodity. The cost matrix need not be square.

An APP was formulated, which can solve any of the three variations of this problem. The formulation of the APP included organization, a description of the command set and timing requirements for the commands. The basic structure of this APP is identical with that of an APP previously formulated for pattern recognition[15,16]. However, it was found that three new features were required for matrix manipulations, namely:

1. A "D" counter capable of being incremented as well as transferring its contents into the data register. It provides row and column indicators for searching purposes.
2. A match indicator DP=1 to indicate when one, and only one word responds to a search operation.
3. A multiple match resolver to select one word from two or more matched words.

These additional features not only give the APP a utility for solving the Hitchcock-Koopmans transportation problem, but also for manipulating matrices required for a wide gamut of problem types. Linear programming, dynamic programming and matrix inversion are examples of possible application extensions.

From the algorithms for the solution of the three variations on the network flow problem, APP type program flow diagrams were constructed. In the case of the binary assignment and general assignment problems, programs in terms of the APP machine language were further written. Since all the basic types of APP operations were defined in these two programs, a machine language program was not written for the transportation problem. From these programs and flow diagrams, solution timing requirements were determined for problems with cost matrices of various sizes.

The solution times for the APP were compared with a serial processor counterpart. In the case of the binary assignment problem, a description of the solution on a particular serial processor was fortunately found in the literature. In the case of assignment and transportation problems, a sophisticated serial processor was hypothesized and an estimate of solution time was derived from the APP flow diagrams.

It was found that the APP has a timing efficiency factor of from one order to three orders of magnitude over its serial processor counterpart for the three variations of the network flow problem considered. The one order of magnitude factor was applicable for smaller matrices considered, of around a 30 × 30 size, and the three orders of magnitude factor was applicable for the largest matrices considered, of around a 1000 × 1000 size where the cost matrix was assumed square for the transportation problem. However, of even greater importance for problems with dynamic data, and where fast solution times are of the essence, as in the case of the weapons assignment problem, the APP determines solutions in a matter of milliseconds, whereas for large matrices the serial processor requires from several seconds up to several minutes. The APP determines solutions in what must be considered as virtual real-time in respect to a human operator's reaction time, which is not the case for serial processors for large matrices.

The weapons assignment problem, representative of network flow problems, was modeled into a Hitchcock-Koopmans transportation problem. However, the Hitchcock-Koopmans transportation problem, and therefore the solutions here presented, have application to problems of interest other than the weapons assignment problem. Other possible applications include:

1 Other types of network flow problems[14], of which one pertinent example is message switching to optimally distribute messages in a complex command and control system.

2. More conventional transportation problems, where the cost matrix might represent estimates of distance or time, as well as cost, and the carriers might include air tankers, ocean tankers, freight cars, trucks or other such vehicles.

3. Personnel assignment problems in which, typically, the problem is to assign N different men optimally to N different jobs.

4. Forms of the traveling-salesman problem[15]. The name comes from the problem of a salesman wishing to travel from his home to each of a number of specified cities, and then return home in such a path as to minimize the total distance (or time). Distribution of material from one common carrier with sequential drop-offs at several destinations is one example of this problem.

REFERENCES - CHAPTER 3

1. Hitchcock, F.L., "The Distribution of a Product from Several Sources to Numerous Localities", Journal of Math. and Physics, Vol. 20, 1941, pp. 224-230.

2. Koopmans, T.C., "Optimum Utilization of the Transportation System", Proc. of the Int'l Statistical Conference, 1947, Washington, D.C., (later reprinted as Supplement to Econometrica, Vol. 17, 1949)

3. Dantzig, G.B., "Programming in a Linear Structure", Comptroller, USAF, Washington D.C., February 1948

4. Dantzig, G.B., "Application of the Simplex Method to a Transportation Problem", Activity Analysis of Production and Allocation, Cowles Commission Monograph 13, 1951

5. Kuhn, H.W., "A Combinatorial Algorithm for the Assignment Problem", Issue 11 of Logistics Papers, George Washington University Logistics Research Project, 1955.

6. Kuhn, H.W., "The Hungarian Method for the Assignment Problem", Naval Research Logistics Quarterly, Vol. 2, Nos. 1 and 2, March-June 1955

7. Egervary, E., "Matrixok Kombinatoruis Tulajdonsagairol", Mathematikai es Tizikai Lapok, Vol. 38, 1931, pp. 16-28 Translated by H.W. Kuhn as Paper 4, Issue 11 of "Logistics Papers", George Washington University Logistics Research Project, 1955.

8. Konig, Dines, "Theorie der Endlichen and Unendlichen Graphen", Akad. Verl. M.B.H., Leipzig 1936 and Chelsea Publishing Company, New York 1950.

9. Ford, L.R., Jr., and Fulkerson, D.R., "A Simple Algorithm for Finding Maximal Network Flows and an Application to the Hitchcock Problem", The RAND Corporation, Paper P-743, September 1955

10. Munkres, J., "Algorithms for the Assignment and Transportation Problems", J. Soc. Appl. Math., Vol. 5, March 1957

11. Manne, A. S., "A Target-Assignment Problem", Operations Research, Vol. 6, 1958, pp. 346-351

12. den Broeder, G. G. Jr., Ellison, R. E., and Emerling, L., "On Optimum Target Assignments", Operations Research, Vol. 7 1959, pp. 322-326

13. Ford, L. R. Jr. and Fulkerson, D. R., "Maximum Flow Through A Network", Can. J. Math, Vol. 8 1956, pg 399-404

14. Heller, S., "The Traveling Salesman's Problem, Part I, Basic Facts", The George Washington University Logistics Research Project, June 1964

15. Fuller, R. H., Bird., R. M., and Worthy, R. M., "Study on Associative Processing Techniques", RADC Report No. TR-65-210 and DDC AD 621 516, August 1965

16. Fuller, R. H., and Bird, R. M., "An Associative Parallel Processor with Application to Picture Processing", Proc. of FJCC, Vol 27, Part I pp 105-116, November 1965

APPENDIX A
AFP SEARCH ALGORITHMS

1.0 INTRODUCTION

This appendix describes a variety of search algorithms which may be used in the Associative File Processor (AFP) of Chapter II.

The associative file processor (AFP) may be visualized as a combination of a rotating disc memory and an associative memory integrated to perform rapid searching of very large data files in response to complex queries.

The data file is stored bit serially on a disc memory having one fixed head for read or write on each track. An associative memory is employed to search simultaneously the parallel bit streams emanating from a number of disc heads against a set of search criteria.

The data base of the AFP consists of a number of files of records stored on the disc tracks. In a typical system each track is divided into 64 blocks, and each block consists of 128 characters with 8 bits per character. The records are made up of fixed-length characters. There will be no loss of generality if it is assumed that one record is 128-characters long, i.e., one record occupies one block on the disc track.

Referring to Figure 1, each record is typically divided into several fields, each of which consists of one or more characters.

Each field to be searched will have one search criterion, consisting of a key word, and a specified search mode. The key word is stored in the key register, and the search mode is stored in the control register which also denotes the fields to be searched. The mask register is used for masking or ignoring some of the bits in the field. In the case of Bounded Search, which requires two key words, the mask register is used to store the other key word.

The search types for individual fields may be designated as follows:

1. Equal to
2. Not equal to
3. Equal to or greater than
4. Equal to or less than
5. Equal to or bounded by

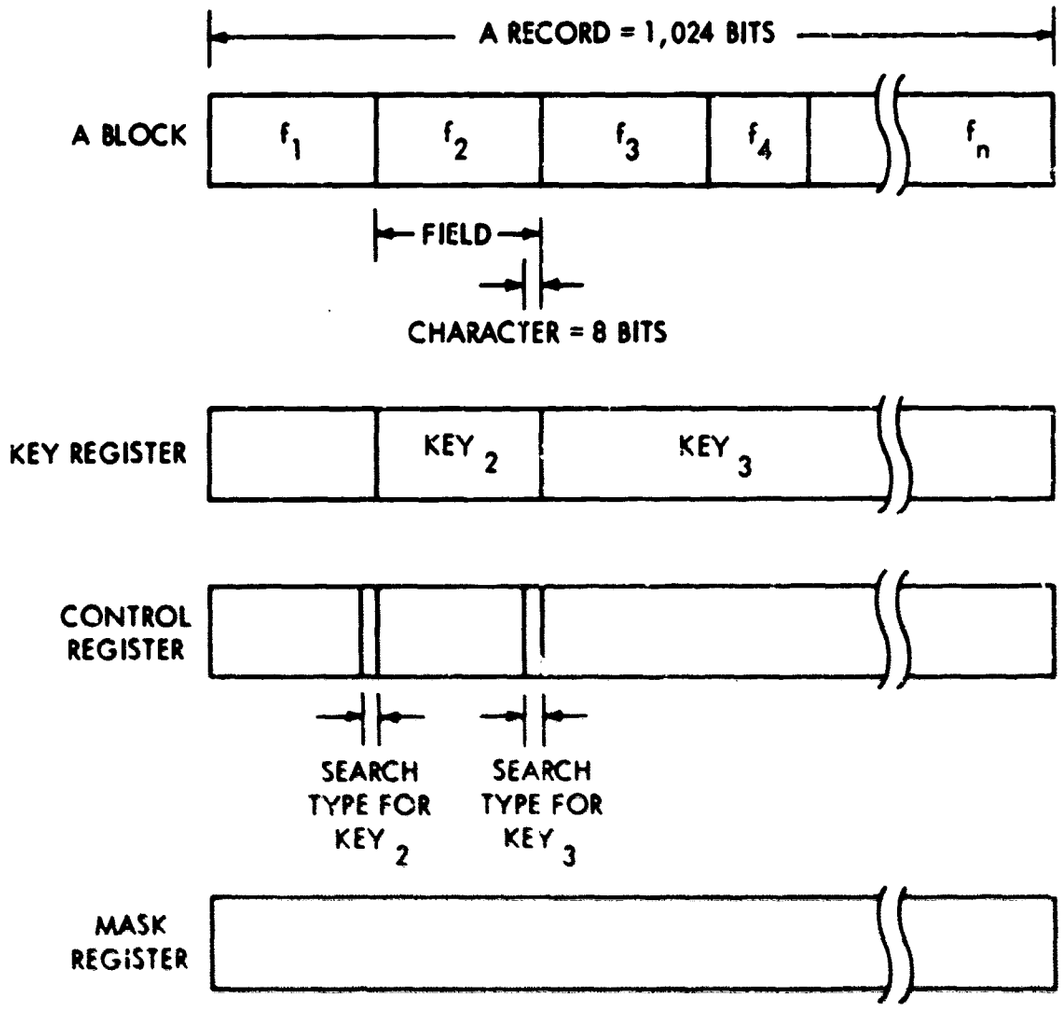


Figure 1 Data Format and Search Registers

Although type 5 is obviously a combination of types 3 and 4, it is included as a separate algorithm because of its frequest use. It is interesting to note that type 2 is the complement of type 1. The word "complement" is used in the sense that type 2 search can be performed by using type 1 and vice versa. Types 3 and 4 are not complementary. The complement of type 3 is "less than", and that of type 4 is "greater than".

For several fields, the search types are designated as follows:

1. Conjunctive search
2. Disjunctive search

The search algorithms with respect to an individual field will be described first, followed by the search algorithms of several fields.

2.0 THE SEARCH ALGORITHMS FOR AN INDIVIDUAL FIELD

2.1 ALGORITHMS USING ONE TAG BIT PER WORD

(a) Equal to

The sequence of bit-by-bit search is immaterial.

Steps:

(1) Set the initial state of the tab bit T_a in each word be "1".

(2) The state of the tag bit T_a is set to be "0", if the bit B_i in the memory word is different from the bit K_i in the key word. In other words, the tag bit T_a is set to be "0", if $B_i = 0$, $K_i = 1$, or if $B_i = 1$, $K_i = 0$.

Otherwise, the tag bit remains the same. It should be noted that once the tag bit T_a is in the "0" state, it remains unchanged there throughout the search sequence.

(3) After the search sequence, the "equal to" criterion is met in memory words with tag bit T_a being "1".

(b) Not Equal to

The sequence of bit-by-bit search is immaterial.

Steps:

(1) Set the initial state of the tag bit T_a in each word to be "0".

(2) The tag bit T_a is set to be "1", if $B_i = 0$, $K_i = 1$ or if $B_i = 1$, $K_i = 0$.

Otherwise, the tag bit T_a remains unchanged.

(3) After the search sequence, the "not equal to" criterion is met in the memory word with tag bit T_a being "1".

The "not equal to" search can be accomplished by using the "equal to" algorithm except that the "not equal to" criterion is met in those words with tag bit T_a being "0".

(c) Equal to or greater than

The sequence of bit-by-bit search must proceed from the least significant to the most significant bits.

Steps:

(1) Set the initial state of tag bit T_a in each word to be "1".

(2) The tag bit T_a is set to be "1", if $B_i = 1$, $K_i = 0$. The tag bit T_a is set to be "0", if $B_i = 0$, $K_i = 1$. Otherwise, the tag bit T_a remains unchanged.

(3) After the search sequence, the "equal to or greater than" criterion is met in the word with tag bit T_a being "1".

It is interesting to note that the "less than" criterion is met in those words with tag bit T_a being "0".

(d) Equal to or less than

The sequence of bit-by-bit search is the same as that in (c).

Steps:

(1) Set the initial state of tag bit T_a in each word to be "1".

(2) The tag bit T_a is set to be "1", if $B_i = 0$, $K_i = 1$. The tag bit T_a is set to be "0", if $B_i = 1$, $K_i = 0$. Otherwise, the tag bit T_a remains unchanged.

(3) After the search sequence the "equal to or less than" criterion is met in the word with tag bit T_a being "1".

It should be obvious that the "greater than" criterion is met in the word with tag bit T_a being "0".

2.2 ALGORITHMS USING TWO TAG BITS PER WORD

(a) Bounded Search

The bits in the key words are represented by M (lower limit) and K (upper limit), so the data field satisfies the criterion:

$$M \leq B \leq K$$

Steps:

(1) The initial states of the tag bits T_a and T_b in each word are both set to be "1".

(2) Tag bit T_a is set to be "1" if, $B_i = 0$, $K_i = 1$, and set to be "0", if $B_i = 1$, $K_i = 0$.
Tag bit T_b is set to be "1", if $B_i = 1$, $M_i = 0$, and set to be "0", if $B_i = 0$, $M_i = 1$.
Otherwise, the tag bits T_a and T_b remain unchanged.

(3) The final states of both T_a and T_b being "1", indicate the "equal to or bounded" criterion is met in the memory word.

3.0 THE SEARCH ALGORITHMS FOR SEVERAL FIELDS

When a number of different types of searches are performed on different fields in a record, it is possible to use only three tag bits per word for the following special cases:

3.1 CONJUNCTIVE SEARCHES

All the different types of searches are to be met, i.e., the "AND" search of

(Type S_0) (Type S_1) (Type S_2) ----- (Type S_n)

where S_0, S_1, S_n can be any of the five types of searches described previously.

Steps:

(1) Set T_c to be "1".

(2) Do search Type S_0, Type S_1 ---- Type S_n. Transfer T_a to T_c, if $T_a = 0$.

(3) Transfer T_b to T_c, if $T_b = 0$.

(4) The final state of T_c being "1" indicates that the conjunctive search is met.

A special case exists if all searches are "equal to" and conjunctive over the fields, which requires only one tag bit.

3.2 DISJUNCTIVE SEARCHES

At least one of the different types of searches are to be met, i.e., the "OR search" of.

(Type S_0) + (Type S_1) + (Type S_2) ---- + (Type S_n)

Steps:

(1) Set T_c to be "0".

(2) Do search Type S_0, Type S_1, ---- Type S_n

(3) Transfer T_a to T_c, if $T_a = 1$.

(4) Transfer T_b to T_a, if $T_b = 0$.

(5) The final state of T_c being "1" indicates the fulfillment of the disjunctive search.

One of the special cases in disjunctive search is that all types of searches are "equal to" which requires only two tag bits.

4.0 LOGIC IMPLEMENTATIONS FOR THE SEARCH ALGORITHMS

The word logic between the disc track and its associative memory word of tag bits is as shown in Figure 2.

The word logic consists of a sense amplifier, a set-reset-complement flip-flop, five gates, and a word driver.

The bit logic functions required by the search algorithms are:

1. Clear the flip-flop each word.
2. Complement the flip-flop in each word.
3. Enable the word driver in each word.
   There are two Enables, Enable W_0 which would cause the word driver to send a current to the memory word such that a "0" may be written at any bit position with bit current present. Similarly, Enable W_1 is for the write of a "1". Of course, the writing can take place only if the flip-flop is in the "1" state.
4. Interrogate the bit in any one bit position. The bit driver would send a current for a NDRO at any one bit position.
5. Enable the bit driver in any one bit position. The bit driver would send a current for a write at any one bit position.
6. Enable the gate G for reading the disc. Disable the gate G for transfer T_a or T_b to any one of the 64 tag bits C_0 through C_{63}.

4.1 ALGORITHMS FOR AN INDIVIDUAL FIELD USING ONE TAG BIT PER WORD

(a) Equal to

Steps: (at the beginning of search)

1. Clear FF in the word logic
2. Complement FF
3. Enable W_1
   Enable D_a

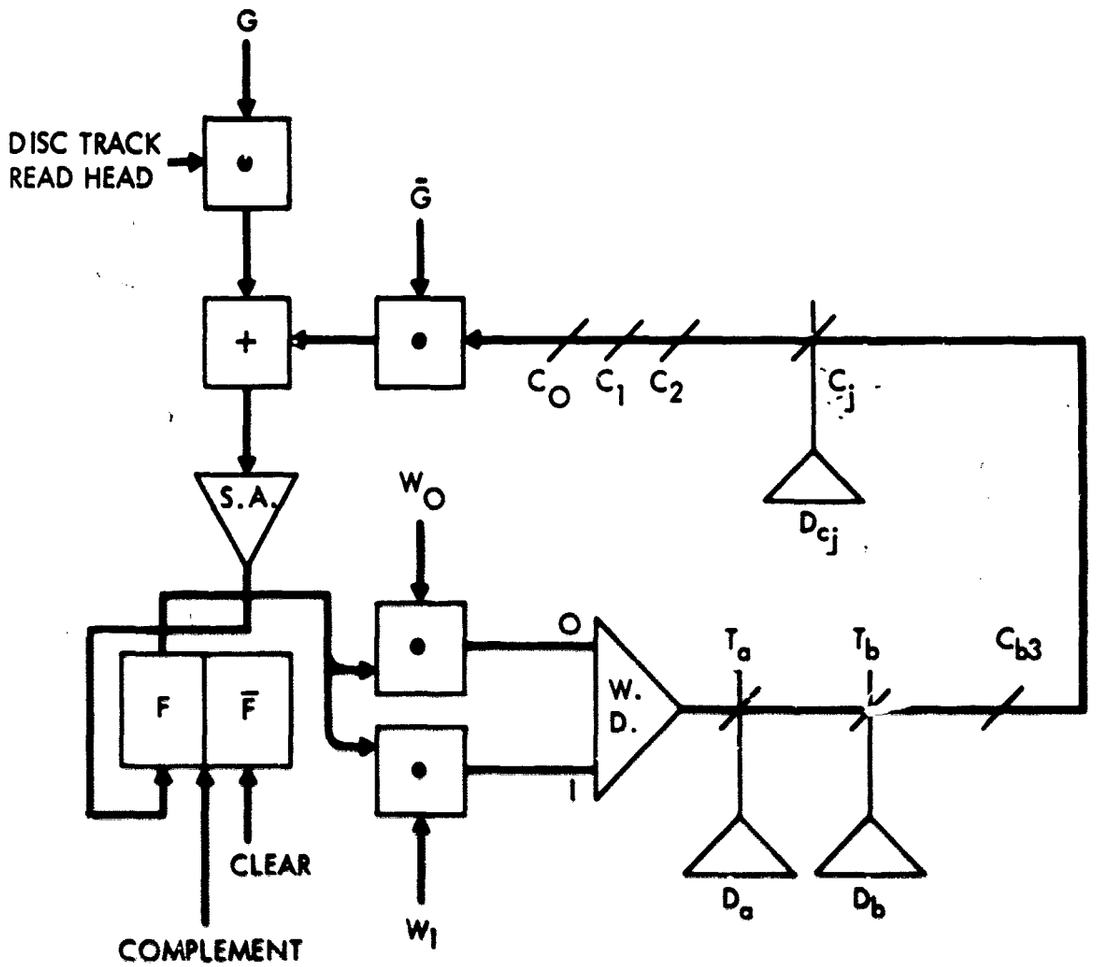


Figure 2 The Word Logic Between Disc Track and Tag Bits Memory Word

Steps: (at each bit positions)

1. Clean FF
2. Interrogate B_i (equivalent to Enable gate-G)
3. Complement FF (if and only if $K_i = 1.$)
4. Enable W_0

   Enable D_a

(b) Not equal to

Steps (at the beginning of search)

1. Clear FF
2. Complement FF
3. Enable W_0
   Enable D_a

Steps (at each bit position)

1. Clear FF
2. Interrogate B_i (equivalent to Enable gate-G)
3. Complement FF (if and only if $K_i = 1$.)
4. Enable W_1
   Enable D_a

(c) Equal to or greater than

Steps (at the beginning of search)

1. Clear FF
2. Complement FF
3. Enable W_1
   Enable D_a

Steps (at each bit position)

1. Clear FF
2. Interrogate B_i (equivalent to Enable gate-G)
3. Complement FF (if and only if $K_c = 1$)
4. Enable W_0 (if and only if $K_i = 1$)
   Enable W_1 (if and only if $K_i = 0$)
   Enable D_a

(d) Equal to or less than

Steps (at the beginning of search)

1. Clear FF
2. Complement FF
3. Enable W_1
   Enable D_a

Steps (at each bit position)

1. Clear FF
2. Interrogate B_i (equivalent to Enable G-gate)
3. Complement FF (if and only if $K_i = 1$)
4. Enable W_0 (if and only if $K_i = 0$)
   Enable W_1 (if and only if $K_i = 1$)
   Enable D_a

4.2 ALGORITHMS FOR AN INDIVIDUAL FIELD USING TWO TAG BITS PER WORD

(a) Equal to or bounded

Steps (at the beginning of search)

1. Clear FF
2. Complement FF
3. Enable W_1
   Enable D_a
   Enable D_b

Steps (at each bit position)

1. Clear FF
2. Interrogate B_i (equivalent to enable G-gate)
3. Complement FF (if and only if $K_i = 1$)
4. Enable W_0 (if and only if $K_i = 0$)
   Enable W_1 (if and only if $K_i = 1$)
   Enable D_a
5. Complement FF (if and only if $M_i = 0$)
6. Enable W_0 (if and only if $M_i = 1$)
   Enable W_1 (if and only if $M_i = 0$)
   Enable D_b

ALGORITHMS FOR SEVERAL FIELDS USING THREE TAG BITS PER WORD

(a) Conjunctive Searches

Steps (to transfer $T_a = 0$ to C_j after each search)

1. Clear FF
2. Interrogate T_a
3. Complement FF
4. Enable W_0
   Enable D_{c_j}

Steps (to transfer $T_b = 0$ to C_j after each search)

1. Clear FF
2. Interrogate T_b
3. Complement FF
4. Enable W_0
   Enable D_{c_j}

(b) Disjunctive Searches

Steps (to transfer $T_a = 1$ to C_j after each search)

1. Clear FF
2. Interrogate T_a
3. Enable W_1
   Enable D_{c_j}

Steps (to transfer $T_b = 0$ to C_j after each search)

1. Clear FF
2. Interrogate T_b
3. Complement FF
4. Enable W_0
   Enable D_{c_j}

5.0 SUMMARY

The foregoing search algorithms can be summarized with two charts as shown in Figure 3 and Figure 4.

Figure 3 shows the states of tag bits T_a and T_b in applying various search algorithms for an individual field.

The mask, key and data bits are represented by M, K and D respectively.

Figure 4 shows the states of C_j in applying various search algorithms for several fields.

The search algorithms have been described by using a minimum of tag bits. However, it should be pointed out that the search time can be reduced if more tag bits are used.

The word logic is normally not necessary in describing the search algorithms, but it may help to understand the algorithms. Furthermore the word logic may demand the change of logical implementation of the search algorithms.

TYPES OF SEARCH	INITIAL STATE OF		CONDITIONS TO SET $T_a = 0$	CONDITIONS TO SET $T_a = 1$	CONDITIONS TO SET $T_b = 0$	CONDITIONS TO SET $T_b = 1$	WHEN SEARCH CRITERION IS MET THE STATE OF	
	T_a	T_b					T_a	T_b
EQUAL	1	NOT IN USE	$MK\bar{D} + M\bar{K}D = 1$	NONE			1	NOT IN USE
NOT EQUAL	0	NOT IN USE	NONE	$MK\bar{D} + M\bar{K}D = 1$			1	NOT IN USE
EQUAL TO + GREATER THAN	1	NOT IN USE	$MK\bar{D} = 1$	$M\bar{K}D = 1$			1	NOT IN USE
EQUAL TO + LESS THAN	1	NOT IN USE	$M\bar{K}D = 1$	$MK\bar{D} = 1$			1	NOT IN USE
BOUNDED	1	1	$M\bar{D} = 1$	$\bar{M}D = 1$	$\bar{K}D = 1$	$K\bar{D} = 1$	1	1

Figure 3 States of Tag Bite T_a and T_b

TYPES OF SEARCH	INITIAL STATE OF C_j	CONDITIONS TO SET $C_j = 0$	CONDITIONS TO SET $C_j = 1$	WHEN SEARCH CRITERION IS MET THE STATE OF C_j
CONJUNCTIVE	1	$\bar{T}_a + \bar{T}_b = 1$	NONE	1
DISJUNCTIVE	0	$T_b = 1$	$T_a = 1$	1

Figure 4 States of C_j

APPENDIX B

DDC TYPE SEARCH IN AFP-2

Described in this appendix are the mechanization and functioning of an associative file processor, specifically tailored to a Defence Documentation Center (DDC, formerly ASTIA) type of problem of document retrieval in response to a priority ordered set of descriptors. The application of an associative file processor (AFP-I) to this problem was discussed in the previous Librascope Study Report (Ref-, Section 2). It is deemed worth while to include the comparative performance of AFPI with that of the processor (AFP-2) described in this report.

A simplified statement of the problem is as follows: Several hundred thousand documents constitute a library from which requests for bibliographies are to be honored. Associated with each document is a short list of applicable descriptors (8 ave., 20 max. in DDC) chosen from a relatively static vocabulary of several thousand defined descriptors (7000 in DDC). A retrieval request is put in the form of a list of descriptors and is matched against the library, evoking matching documents.

In practice, a problem arises in that the retrieval request may be too general or too specific, i.e., it may evoke too many documents, or too few (e.g., none). Consequently, a priority is assigned to the descriptors in the retrieval request, and they are used in sequence to narrow the set of evoked documents to an acceptable number. A

relevancy level of the document descriptors in the library is also utilized, but this is a minor complication and will not be treated here.

In applying an associative file processor to the retrieval problem, the first consideration is that in a document's descriptor list, a given discriptor may appear anywhere, i.e., first, second or last. Fortunately, with a well defined descriptor vocabulary, descriptors can be encoded into fixed length fields, and, since there is a maximum number allowable to any document, each document can be represented by a fixed length record.

The document retrieval process is basically a conjunction of disjunctive searches, as referred to in section 2.3.8.2. That is, the search is conducted as follows: The first retrieval descriptor is loaded into every possible field of the key register. Then a search is performed, leaving compare flags set if any field matches (disjunction). Then the next descriptor is loaded and treated similarly. The search then leaves those compare flags where some field matches the second descriptor (conjunction of disjunctions). Actually, the mechanization may be thought of more simply as follows: The compare flags are initially all set on. Then each search sets the compare flags off for those records where the descriptor was not found (regardless of position). Thus, the

number of compare flags is reduced with each search, until an acceptable number of documents are flagged.

As was discussed in Section 2.3.8.2, there is a tradeoff of associative memory size and retrieval speed. That is, the most cost effective system may well search only a fraction of the total disc memory per revolution, and take correspondingly many revolutions per descriptor search. It is possible, in this case, to store compare flags on the disc, after each descriptor search, rather than in the associative memory, as noted in section 2.3.8.2. On the other hand, the associative memory word length could be expanded to accommodate all of the compare flags. However, this seems unlikely to be economical in view of the per-bit cost of associative versus disc memories.

The difference in performance between AFP-I and AFP-2 is dependent on the following factors: AFP-I searched the whole disc memory content on all retrieval descriptors in one revolution. The AFP-2 requires k d revolutions, where 1/k of the disc tracks are searched at once, and there are d retrieval descriptors. However, offsetting the speed disadvantage of AFP-2, is the fact that the compare flag count is available at each stage of the search, so it can be stopped as desired, and an efficient readout of qualifying document numbers is possible. Neither of these advantages was a feature of the AFP-I design.

APPENDIX C

A WOVEN PLATED-WIRE ASSOCIATIVE MEMORY

By: T. H. Fuller, J. C. Tu and R. M. Bird
General Precision Inc, Librascope Group

1.0 INTRODUCTION

This paper describes an organization and discusses circuits for an associative (parallel search) memory using woven plated-wire memory elements, as initially developed by TOKO, Inc. of Tokyo, Japan, and now available at Librascope. This design allows fabrication of moderate-capacity airborne associative memories having high processing speeds together with small size, weight, power and cost. The memory is operable over a wide temperature range. Highly iterated word electronics are realized economically through the use of integrated circuits and magnetic elements. Novel design features include the use of batch-fabricated plated-wire memory elements, bit-parallel readout from an associative organization of these elements, and the use of magnetic cores in word logic to allow simultaneous writing into selected words. Suitable tasks for such associative memories are discussed. Important characteristics of the Librascope associative memory are as follows:

1) Stored data may be located on the basis of content which is evaluated simultaneously over all stored words.
2) Memory word content may be specified to be equal to, greater than, or less than a key word.
3) Responding words (i.e., words satisfying a search) may be read out or may be rewritten in a bit-parallel mode.
4) For multiple responses, words may be accessed sequentially for reading or writing.
5) All responding words may be rewritten simultaneously in some or all bits (i.e., multiwrite).

The efficiency of associative memory techniques is closely dependent upon the characteristics of the memory elements used for their implementation. The increased logic complexity of associative memories, over conventional coordinate addressed memories, will lead to excessive system cost unless inexpensive high-performance elements are used in the memory array. The Librascope associative memory array consists of permalloy-plated copper wires woven together with insulated copper wires to form a wire mesh matrix as shown in figure 1. Groups of insulated wires are interconnected to form multi-turn digit coils. Bits are stored in permalloy-plated wires at each intersection with a word coil. Plated wire is a form of magnetic film memory whose advantages have been widely recognized, [1, 2, 3, 4, 5] but has only recently become available to the memory system designer. These plated-wire elements are of interest as associative memory arrays for the following reasons:

1) Stored information may be non-destructively read, generating search output voltages which are significantly larger than have been obtained with other film memories. Highly iterated word-sense amplifiers may thus be realized as integrated circuits.
2) Word currents, which may be required with high multiplicity during multiwrite, are small. In our design, these currents are simply generated by magnetic cores set from word-sense amplifiers.
3) Writing in both conventional and multiwrite modes is much faster, by a factor of 10 to 30, and requires less power than for other known non-destructive memory elements.
4) The memory may be destructively read in a conventional bit-parallel word-serial mode.
5) Array cost is now much less, by a factor of 10, than for discrete element arrays and will decrease as woven planes are produced in volume (perhaps to a factor of 100).

In Section 2 of this paper, we review the characteristics of plated-wire elements and arrays and characterize arrays for associative applications. In Section 3, we describe an organization and a command set for a plated-wire associative memory. Memory circuit considerations are discussed in Section 4. In the concluding section, plated-wire associative memories are compared with competing realizations and potential applications for these memories are discussed.

2.0 PLATED WIRE MEMORY ELEMENTS AND PLANES

2.1 MEMORY ELEMENTS

The woven memory plane uses a memory element in which permalloy film is plated on a copper wire with current flowing in the wire to create a circumferential easy axis of magnetization. The copper substrate serves as both a digit drive and sense line for information stored in the permalloy film. The flux path is closed for the easy axis in such a manner that demagnetization does not occur and coupling to the digit drive-sense line is high. The permalloy film switches in a rotational mode, and thus allows extremely rapid reading and writing. The combination of shape anisotropy, due to cylindrical geometry of the film and the field induced anisotropy introduced in plating, gives small skew and dispersion from the [illegible] easy axes, and thus provides a good operating margin for non-destructive readout. Batch-fabrication techniques are used both in plating the wire and in weaving the planes.

The information state of a bit cell is non-destructively sensed by passing a current

through a coil of drive wires orthogonal to the plated wires. This current establishes an axial field, less than the anisotropic field H_k, and rotates the magnetization vector into a hard direction by an angle less than 90°. The sign of voltage induced in the plated wire indicates the original circumferential flux direction of the film (figure 2). When coil current is removed, the magnetization vector returns to its initial state, inducing a voltage of opposite polarity in the plated wire. Information is written into the wire by coincidence of an axial field, less than H_k, with a circumferential field less than H_c. The final information state of the film is determined by the direction of the circumferential field; hence by the direction of the digit current flowing in the plated wire.

2.2 MEMORY PLANES

To form woven memory planes, plated wires are woven together with insulated drive lines into a cloth-line mat, using equipment similar to ordinary textile weaving equipment. Plated wires run in the weft direction of the weave; insulated wires run in the warp direction. The mat is then mounted on a printed circuit board to form a memory plane. TOKO Inc., in cooperation with Kokusai Denshin Demiva Company, perfected both plating processes and weaving processes for this type of memory and TOKO now has production capability for woven planes.

Librascope will manufacture memory planes and systems for sale in the United States. We have designed and experimentally verified several memory systems using TOKO planes, and the resulting evaluation has pointed the way toward the next generation of improved planes and memory systems. These systems include a submicrosecond NDRO memory[5] and the associative memory reported herein. Our evaluation of woven planes shows that the weaving technique offers advantages in each of these systems over techniques using simple strip drive lines in the following areas:

1) Density of bits along a plated wire may be somewhat increased without interference between adjacent bits. This is true because properly designed woven drive lines allow a faster fall off in fringing drive field than do striplines. Packing densities of 25 bits per inch have been achieved in woven planes as compared to 20 bits per inch reported for strip line planes

2) Word drive current requirements are lessened by the use of a multiturn word line.

3) Cost of manufacture appears less, due to the availability of more highly automated plane production techniques. TOKO coil has successfully expended considerable production engineering effort to realize cost reductions possible with weaving techniques.

Electrical specifications for a typical woven memory plane are shown in Table 1.

TABLE I

Specifications for a Woven Memory Plane

Word Write Current[1]	I_{ww}	50 ma
Word Read Current[1]	I_{wr}	100 ma
Digit Write Current[1]	I_{dw}	400 ma
Digit Read Current[1]	I_{dr}	150 ma
Word-Line Sense Voltage[2]		5 mv
Bit-Line Sense Voltage		5 mv

[1]This notation applies to associative organization of the planes as in figure 1.

[2]Rise time for interrogate current I_{dr} is 200 ns.

2.3 ASSOCIATIVE ARRAYS

A practical requirement for associative arrays is the ability to search the array without disturbing its information content. The response of each memory word to a search must be separately distinguishable. Memory elements must, therefore, be capable of NDRO operation with NDRO outputs appearing on word lines. Since NDRO outputs appear on plated wires in woven planes, plated wires must serve as word lines in an associative array (see figure 1). This contrasts with a conventional coordinate-addressed organization in which plated wires serve as digit lines.

Equality search is implemented in a word-parallel, bit-serial fashion as illustrated in figure 3. A search key and four stored words, W_1 - W_4 are shown in figure 3a. Digit currents (figure 3b), corresponding to "1's" in the key, are turned on simultaneously during a presearch period. During search time, digit currents I_{D1} - I_{D4} are changed sequentially (i.e. currents are turned on if initially off and conversely). Recall, (figure 2), that bits storing "0" have negative outputs during the fall of the reading current. A word which mismatches the key in some bit position will have a negative output voltage c_w, when the mismatching bit is searched. For words matching the key (word W4 of figure 4), c_w is positive at all bit search times. The described search algorithm thus implements the "exclusive or" function required for equality search. Several algorithms are available to identify stored words larger (smaller) than a key. These algorithms have been well described in the literature[6, 7] and will not be repeated here.

It is desirable that words responding to a search be read in bit-parallel fashion. In this reading mode, the insulated digit wires act as sense lines. This reading mode may be destructive since words are read sequentially and may be rewritten after reading. A novel sensing method was developed to meet this requirement.

All plated word lines are biased by a small hard-axis field. A word-read current (figure 4), sufficient to cause incoherent rotation of the film, is applied to a selected word line. Bits initially in the "1" state switch to the zero state, inducing a voltage into the digit line. Bits initially in the "0" state are not changed and induce only a small voltage into the digit line. The word is rewritten by applying a smaller current of opposite polarity to the word line. This current, in coincidence with digit currents on selected digit lines, restores selected bits of the word to the "1" state.

3.0 ORGANIZATION AND COMMAND SET

The associative memory is organized as shown in figure 5. The data register buffers communication between this memory and other system components. The mask register indicates that portion of data register contents which constitutes search criteria. The detector plane stores results of searches. The match resolver allows sequential access to a multiplicity of matching words. The match indicator denotes the presence or absence of some matching word. A data word may be written or read through the data register. Selected bits of many data words may be rewritten simultaneously by a multiwrite command. Equality searches are implemented in a bit-serial mode.

A simplified 3-word by 4-bit search memory array is shown in figure 1. Bits of each word are stored along one plated wire, being defined at each intersection of a plated wire with a digit coil. Each word has an associated detector element containing a sense amplifier, a match storage element, and circuitry for resolution of multiple matches. Each word also has an associated bipolar word driver which, in coincidence with digit current, writes elements to the "1" or "0" state. For NDRO reading, digit current I_D is supplied and sense outputs are read from the plated-wire word line.

3.1 COMMAND LIST FOR ASSOCIATIVE MEMORY

Equality Search: Digit currents are first established at all unmasked "1" digits in the search key. Digit currents are then altered serially (i.e., turned on for a "0" and off for a "1") for all unmasked key bits. Digit drivers for "0's" in the key are then turned off. Mismatching words are indicated by negative voltages in plated wires (figure 4), which set initially cleared detector elements to the mismatched state. The match indicator then denotes whether any match exists. If several matches exist, these may be accessed sequentially by use of the match resolver described in Section 4.

Multiwrite: This command writes simultaneously into selected bits of all matching words. Two methods for multiwriting will be described. For the first method, digit currents are established simultaneously at all bit columns which are to be written to "1". In coincidence with these currents, "1" writing word currents are established at matching words. Only those bits subject to both currents switch to the "1" state. The process is repeated using "0" writing word currents to write selected bits of matching words to the "0" state.

Operating margins are improved if all bits of a column are rewritten simultaneously. Writing must now be serial by column over all altered columns. Prior to writing a column, its contents are read to the detector plane by a single column search. At matching words, new data are inserted into detector elements, and the entire column is then rewritten. This method is more time-consuming and requires some complication of word electronics relative to the former method, but it requires less-stringent control of memory plane characteristics.

Word Read: This command reads matching words, sequenced by the match resolver, into the data register. Execution of this command requires bit-oriented sense lines as well as the word-oriented sense lines used for match detection. A bit-parallel destructive readout is obtained by passing a current of approximately 100 ma down plated word lines. This current switches all bits on this line to the "0" state and, for bits previously in the "1" state, couples signals in excess of 5 mv to the non-plated bit lines. This technique greatly simplifies reading as compared with previously known methods which require bit-serial reading of a selected word through the plated word line. The word read command may be given repeatedly following a single search to read sequentially all matching words, each selected by the match resolver.

Word Write: Contents of the data register, in bit positions unmasked by the mask register, are written into some word, selection on the basis of a previous search, by match resolution circuitry. Word contents are unchanged in masked bit positions, thus allowing selective alteration of multiple fields stored in a single word.

The word is first cleared to "0" by a word read command. Data in masked fields of the word are read into the data register for rewriting. New data are inserted in unmasked fields of the data register. Simultaneously, a word current is established with a polarity to write "1's" in the selected word. The word write command must be preceded by a word read command.

4.0 CIRCUITRY

A discussion of circuitry in this paper is limited, because of space constraints, to two of the most critical and costly circuits in the associative memory, the word circuitry and the match resolution circuitry.

4.1 WORD CIRCUITRY

Figure 6 illustrates the logic diagram for the word electronics. The flip-flop is initially

cleared to the "1" state. Any serially sequenced mismatch on the bit line will reset the flip-flop to the "0" state. At the conclusion of the search operation, word flip-flops for matching words remain in the "1" state.

A novel word logic, shown in figure 7, uses the two magnetic cores, as shown, to replace the flip-flop, the two "AND" gates and the word driver shown in figure 6. The operation is as follows:

1) The output stage of the sense amplifier is a latching switch. It is initially cleared.

2) Simultaneously with operation (1), cores "0" and "1" are reset in the direction of the arrows by the reset switch common for all words.

3) The sense amplifier is strobed to eliminate spurious signals occuring at times other than a search cycle.

4) On any one mismatch, cores "0" and "1" will be set in a counterclockwise direction.

5) After the search cycle, a "1" or "0" may be written for matched words by setting either the set "1" or set "0" switch respectively which are common for all words.

This magnetic logic substantially reduces complexity of word circuitry and thus cost of an associative memory system.

4.2 MATCH RESOLUTION CIRCUITRY

This circuit sequentially addresses matched words in the memory so that each can be read or written into.

One type of match resolution circuit, representing a compromise between high speed and cost is the two-dimensional type shown in figure 8. The match resolution circuit is implemented by a matrix of magnetic cores which are set by the matched word outputs. Each core has five threaded wires; one from a particular memory word, one each from the row driver and row register, and one each from the column driver and column register. The operations are as follows:

a) Before an associative memory search operation is started, all matrix cores and column registers are cleared to the matched state.

b) After the search operation, all mismatched words set their appropriate cores to the mismatched state. In turn, these cores set their corresponding column registers to the mismatch state.

c) Next, either sequential or high-speed, tree-type logic networks are used to select the first matched column register (starting from R_1) which was set by the core or cores in the corresponding column during the operation in (b).

d) All row registers are then cleared to the mismatch state.

e) Next, a half select current is sent down both the selected column line and all row lines, setting those row registers corresponding to matched cores for all rows in the matched column.

f) Next, as with the column registers, either sequential or high-speed, tree-type logic networks are used to search through the row registers selecting matched rows. Knowing a matched row for a particular matched column, then, determines the location of a matched word.

g) Steps (c) through (f) are repeated by selecting the next matched column until all matched column registers have been found.

5.0 CONCLUSIONS

Associative computing methods have found little use to date, due to their well-nigh prohibitive cost. Nevertheless, their applications are many and await only a significant reduction in the cost per storage bit as well as a simplified realization of the word electronics. Plated-wire associative memories afford a reduction in cost per storage bit by a factor of the order of 20 to 1. The magnetic logic implementation for the word electronics, which has been presented, permits approximately a 10-to-1 cost reduction for these circuits. These advantages permit the development engineer to take a fresh look at associative computing methods as solutions to a wide range of data processing problems.

Plated-wire realizations afford advantages, other than cost reduction. They provide large NDRO outputs and employ small word write currents. Writing selected bits into many words simultaneously is possible because of the small write power requirements. The importance of this multiwrite as an associative processing operation has been stressed by several investigators[8, 9]. Both write times and memory search times are significantly faster than for discrete magnetic elements.[10]

Librascope has extensively studied associative processing techniques, both under contract from RADC and under independent company funding. Our studies and those of others show associative processing techniques to be useful in visual pattern recognition[8, 9, 11], in solution of partial differential equations[8], in Elint pulse train separation, and in a variety of information retrieval systems[12]. Use of these techniques is indicated whenever some sequence of computer arithmetic or logic commands is to be executed independently over many sets of data elements. The parallelism, inherent in associative processing, allows the command sequence to be executed simultaneously over all data sets. The speed gain over conventional sequential techniques can be very large if a high degree of parallelism is allowed by the algorithm chosen for problem solution. Such is the case in all problems referenced above.

REFERENCES

1) Futami, K., S. Oshima, and T. Kamibyashi, "The Plated-Woven Wire Memory Matrix", Proc. Intermag Conference, April 6-8, 1964 Washington, D. C.

2) Danylchuk, I., Et al, "Plated Wire Magnetic Film Memories", Proc. Intermag Conference, Washington, D. C., April 6-8, 1964

3) Maeda, H., et al, "Woven Thin Film Wire Memory", Proc. Intermag Conference Washington, D. C., April 6-8, 1964

4) Bartik, W. J., et al, "A 100 Megabit Random-Access Plated Wire Memory", Proc. Intermag Conference, Washington, D.C., April 6-8, 1964

5) Bienhoff, M., et al, "Considerations in Design of Plated Wire Memory Systems", Symposium on Impact of Batch Fabricaion on Future Computers, Los Angeles, April 6-8, 1964

6) Fuller, R. and G. Estrin, "Algorithms for Content Addressable Memory Organizations", Proceedings of the Pacific Computer Conference, Pasadena, March 15-16, 1963

7) Falkoff, A.D., "Algorithms for Parallel Search Memories", Journal of the ACM, Vol 9, pp 488-511, October 1962

8) Fuller, R., and G. Estrin, "Some Applications for Content Addressable Memories", Proc. FJCC, Las Vegas, November 1963

9) "Computer Associative Memory Study", Final Report on Contract AF 04(695)-318, Space Technology Laboratories, July 15, 1964

10) "Research on Ferret Associative Memory" Interim report on Contract AF 33(615)-1259, Philco Corp., November 1964

11) "Associative Processing Techniques," Interim Report on Contract AF 30(602)-3371, Librascope, August, 1964

12) "Associative Processing Techniques" Final Report on Contract AF 30(602)-3371, Librascope, May 1965.

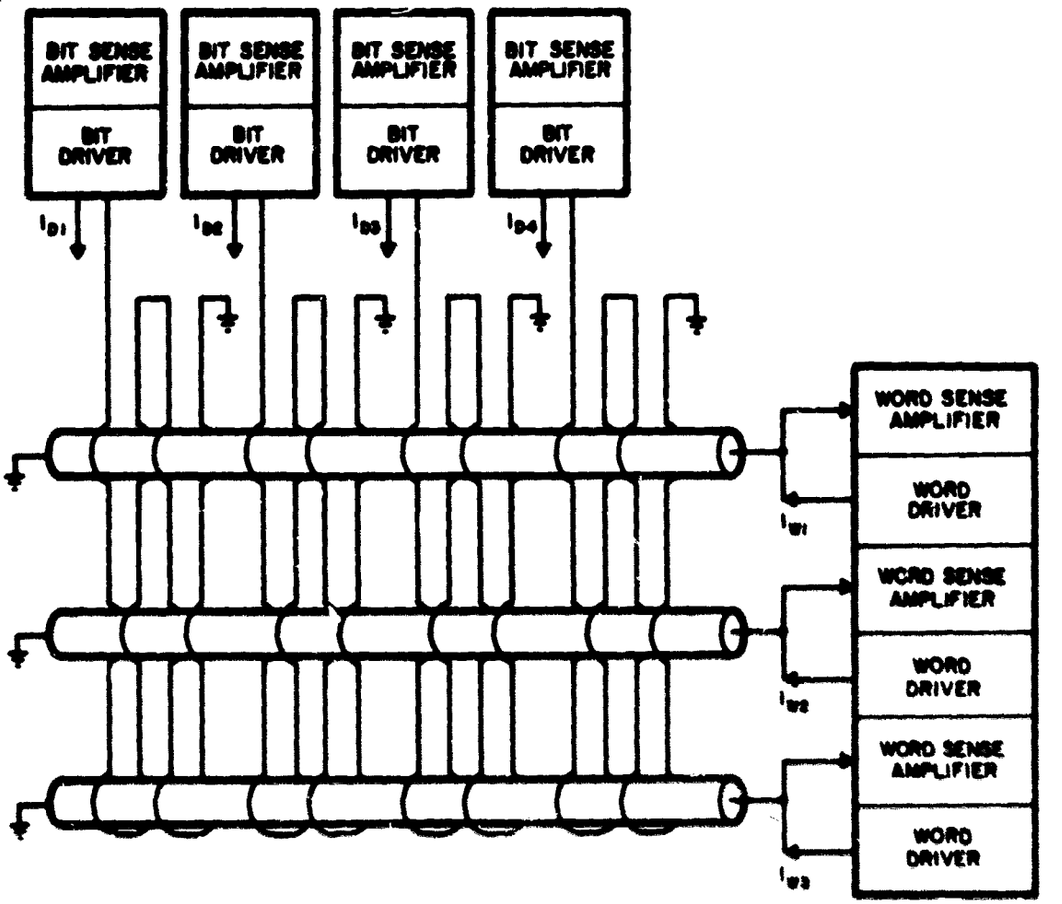
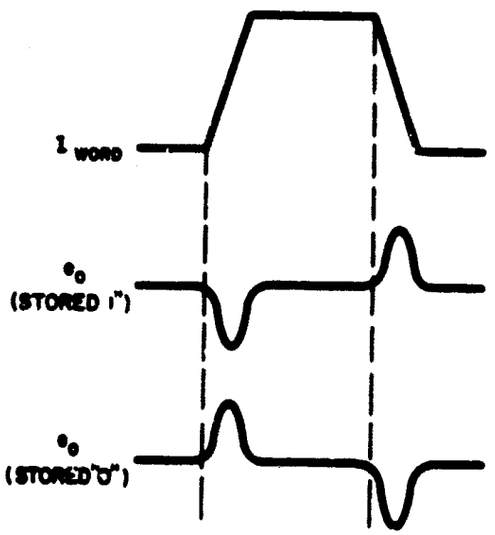


Figure 1. Associative Memory Array

A. READ



B. WRITE

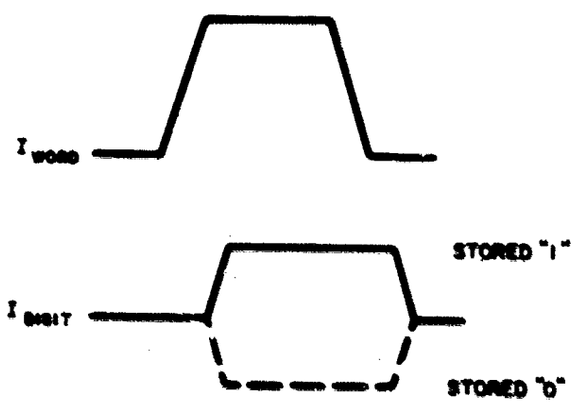
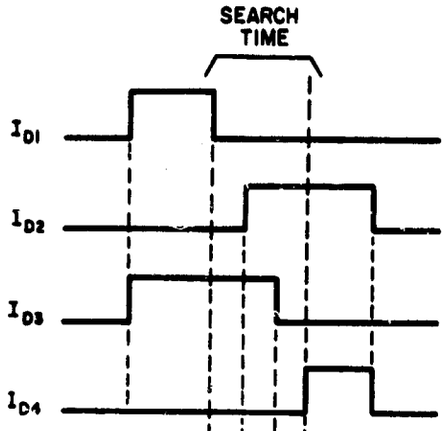


Figure 2. Reading and Writing of Plated-Wire Elements

A.

	D1	D2	D3	D4
KEY	1	0	1	0
W1	0	0	0	0
W2	1	1	1	1
W3	0	1	0	1
W4	1	0	1	0

B.



C.

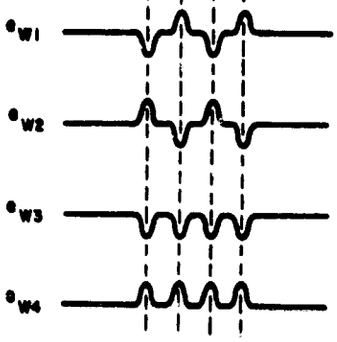


Figure 3. Equality Search

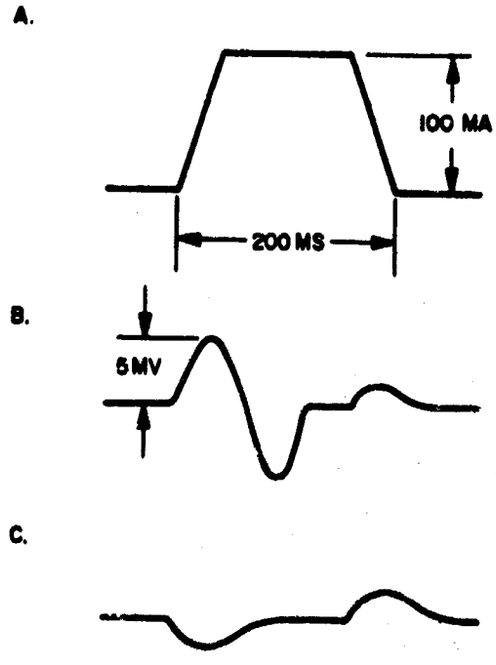


Figure 4. Destructive Readout

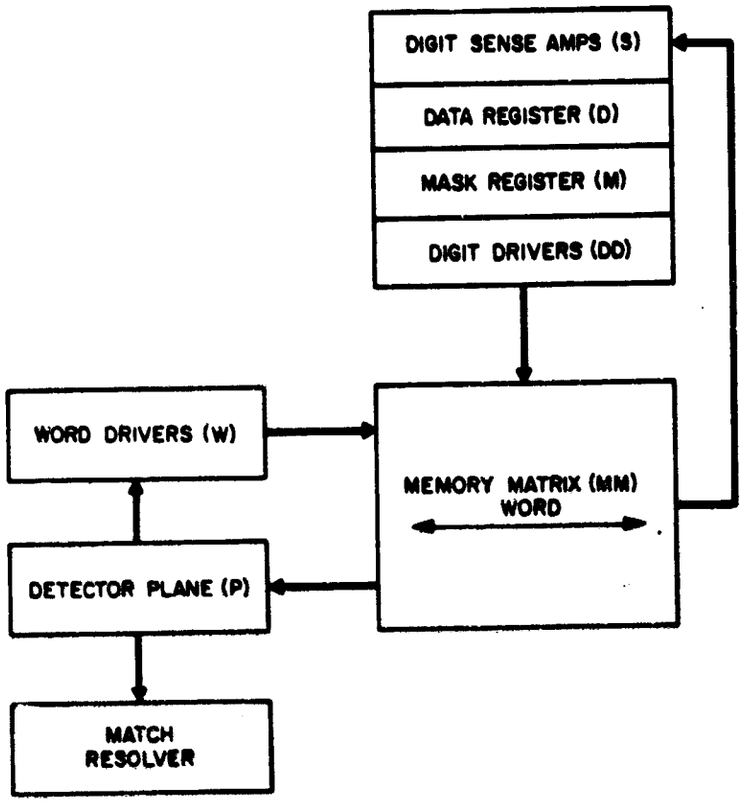


Figure 5. Block Diagram for Associative Memory

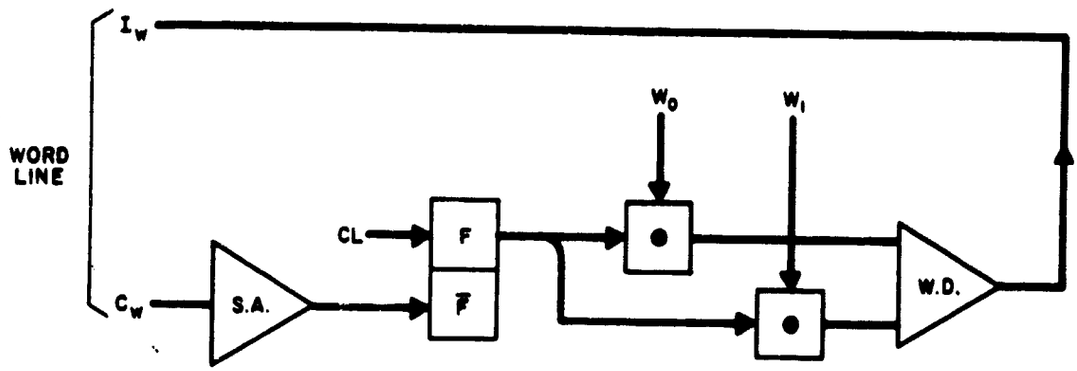


Figure 6. Word Logic Requirements

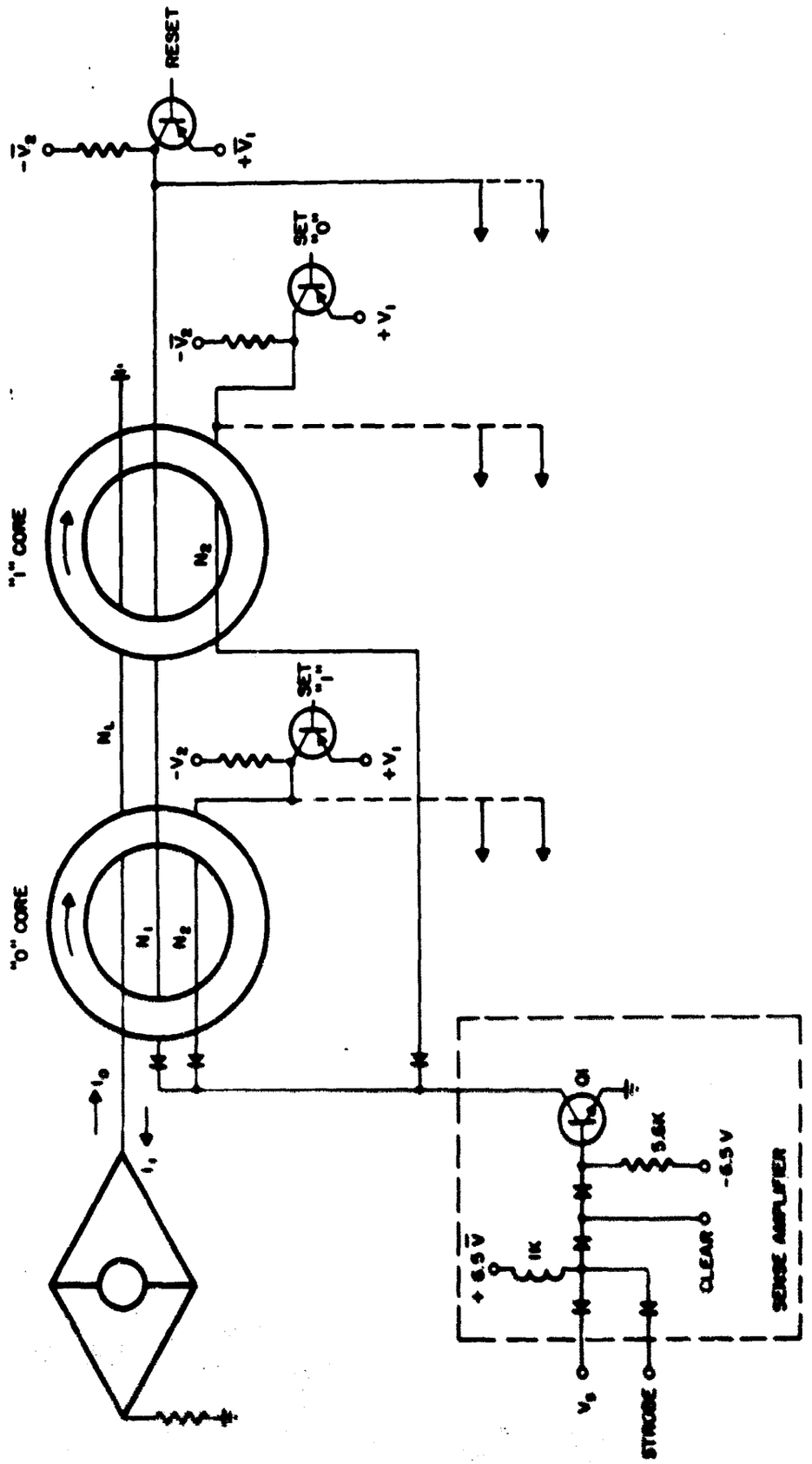


Figure 7. Core Word Logic Implementation

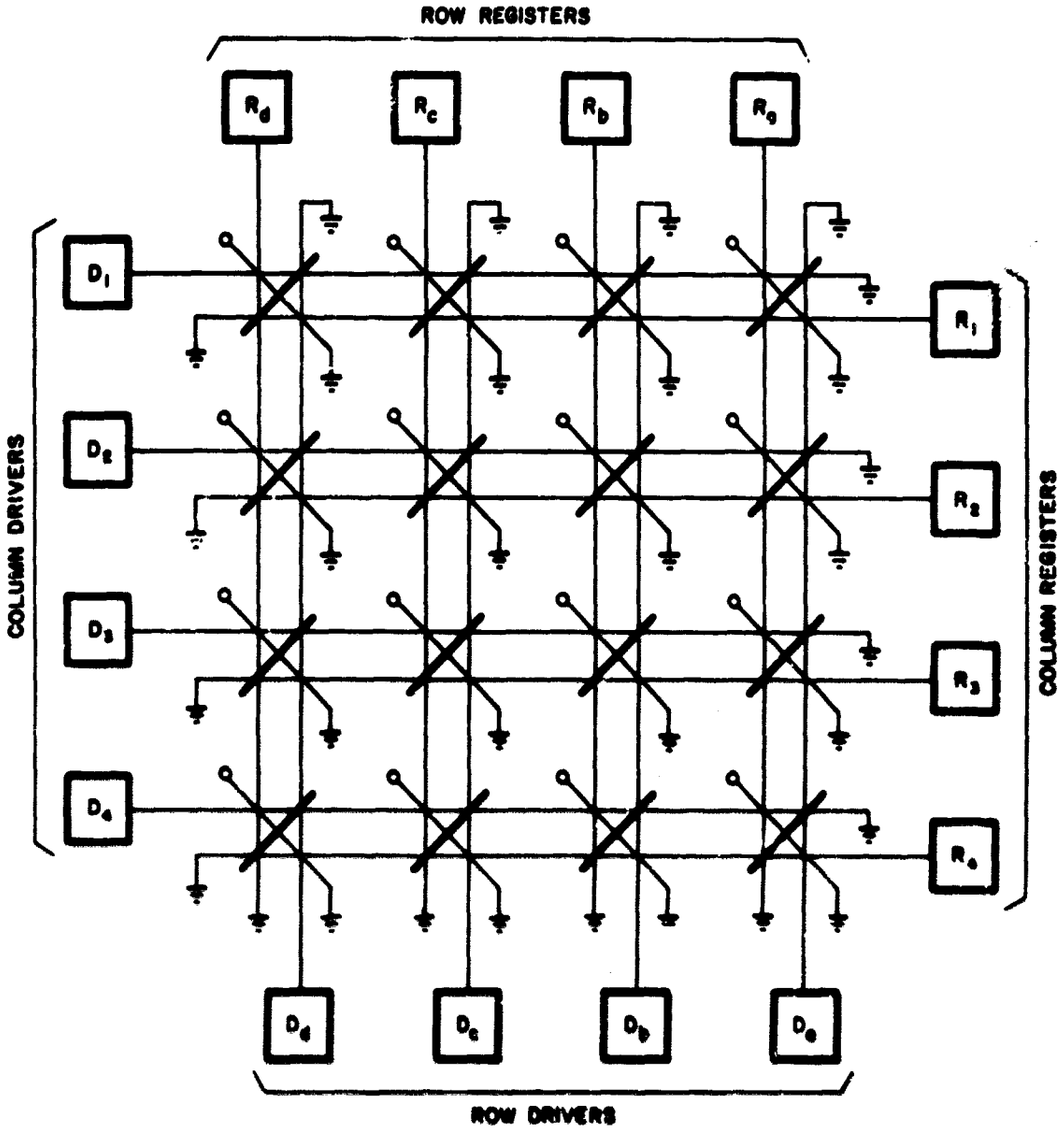


Figure 8. Two-Dimensional Match Resolution Circuit

APPENDIX D

PROGRAM IN MACHINE LANGUAGE CONTROL

FOR GENERAL ASSIGNMENT PROBLEM

A-1 ——→ A-56 INITIAL ADJUSTMENT FOR MINIMIZING

	DP	A	B	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	D	
A-1	Set D = P max												sets row and column counter
A-2	Set A_{FL} = d_i max, A_{IL} = d_i min												
A-3	Set B_{FL} = i max, B_{IL} = i min												
A-4	Transfer D into DR under B counter control												
A-5	1	S O $\bar{W}$ L	S D $\bar{W}$ 1.	- - -	- - -	- - -	- - -	- - -	- - -	- - -	- - -	1	Search $D^2 = 0$, i = p, Increment D
A-6	IF DP ≠ 0 jump to A-29												
A-7	1	- - - -	- - - -	$\bar{S}$ I W	- - -	- - -	- - -	- - -	- - -	- - -	- - -	-	Clear T_1 = 1
A-8	Set A_{FL} = d_i min, A_{IL} = d_i max												
A-9	0	S O $\bar{W}$ N	S D $\bar{W}$ L	S I $\bar{W}$	- - -	- - -	- - -	- - -	- - -	- - -	- - -	-	Search D_d = 0, T_1 = 1, i = p
A-10	IF DP = 0 jump to A-13												
A-11	IF DP = 1 jump to A-18												
A-12	1	S I $\bar{W}$ N	S D $\bar{W}$ L	S I W	- - -	- - -	- - -	- - -	- - -	- - -	- - -	-	Search D_d = 1, T_1 = 1, i = p. Write T_1 = 0
A-13	1	- - - D	- - - -	- - -	- - -	- - -	- - -	- - -	- - -	- - -	- - -	-	

	DH	A	B	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	D	
A-14	IF A counter ≠ initial limit, jump to A-9												
A-15				S I W									Search $T_1 = 1$ Write $T_1 = 0$
A-16	N	Clear and select 1st match											
A-17	N			$\overline{S}$ O W									Write $T_1 = 1$
A-18	Set AFL = d_i max, AIL = d_i min												
A-19						$\overline{S}$ I W							Clear $T_3 = 0$
A-20		S O $\overline{W}$ N		S I $\overline{W}$									Search $D_d = 0$, $T_1 = 1$
A-22	IF DP ≠ 0, jump to A-26												
A-23		S O W N	S D $\overline{W}$ L			S I $\overline{W}$							Search $D_d = 0$, $T_3 = 1$, $i = p$, Write $D_d = 1$
A-24		S I W I	S D $\overline{W}$ L			S I W							Search $D_d = 0$ $T_3 = 1$, $i \neq p$ Write $D_d = 0$, $T_3 = 0$
A-25	Jump to A-26												
A-26		S I W N	S D $\overline{W}$ L			S O $\overline{W}$							Search $D_d = 1$ $T_3 = 0$, $i \neq p$ Write $D_d = 0$
A-27		S O W I	S D $\overline{W}$ L			S O W							Search $D_d = 0$ $T_3 = 0$, $i \neq p$ Write $D_d = 1$
A-28	IF A counter ≠ initial limit, jump to A-20												

	DP	A	B	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	D	
A-29	IF D / 1 jump to A-5												
A-30	Set B_{FL} = j max, B_{IL} = j min												
A-31 ↓ A-56	Same a A-5 through A-29 with jumps to approximate counterparts												
	A-57 ⟶ A-64, INITIAL INDEPENDENT ZERO ASSIGNMENT												
A-57	I	- - - -	- - - -	S I W	- - -	- - -	S I W	- - -	- - -	- - -	- - -	-	Clear $\overline{T_1}$ = 0, T_4 = 0
A-58	I	S O $\overline{W}$ L	- - - -	S O $\overline{W}$	- - -	- - -	- - -	- - -	- - -	- - -	- - -	-	Write T_1 = 1 for d = 0
A-59	Set B_{FL} = i max, B_{IL} = i min												
A-60	Set A_{FL} = j max, A_{IL} = j min												
A-61	Transfer D into DR under B counter control												
A-62	I	- - - -	S D $\overline{W}$ L	S I $\overline{W}$	- - -	- - -	S O $\overline{W}$	- - -	- - -	- - -	- - -	I	Search i = p, T_1 = 1, T_4 = 0
A-63	IF DP = 0, jump to A-69												
A-64	IF DP = 1, jump to A-66												
A-65	N	Clear and select first match											
A-66	N	- - - -	- - - -	- - -	$\overline{S}$ O W	- - -	- - -	- - -	- - -	- - -	- - -	-	Write T_2 = 1 (starred zero)
A-67	Transfer matched word into DR												

	DH	A	B	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	D	
A-68	1	S D $\bar{W}$ L	- - - -	- - -	- - -	- - -	$\bar{S}$ O W	- - -	- - -	- - -	- - -	-	Search q = DR, write, T_4 = 1

A-69 IF D ≠ 1, jump to A-61

A-70 → A-83 COST MATRIX TEST

A-70	1	- - - -	- - - -	- - -	- - -	$\bar{S}$ 1 W	- - -	$\bar{S}$ 1 W	- - -	- - -	- - -	-	Clears T_1 = 0, T_4 = 0
A-71	1	- - - -	- - - -	S 1 $\bar{W}$	- - -	- - -	S O $\bar{W}$	- - -	- - -	- - -	- - -	-	Search T_1 = 1, T_4 = 0

A-72 IF DP = 0, jump to A-84

A-73 IF DP = 1, jump to A-75

A-74 N Clear and select 1st match

A-75	N	- - - -	- - - -	- - -	- - -	- - -	- - -	S O W	$\bar{S}$ O W	- - -	S O W	-	Write T_5 = 0, T_6 = 0, T_7 = 0

A-76 Transfer matched word into DR

A-77	1	- - - -	S D $\bar{W}$ L	- - -	S 1 $\bar{W}$	- - -	- - -	- - -	- - -	- - -	- - -	-	Search i = DR, write T_2 = 1

A-78 IF DR = 0, jump to A-117

A-79	1	- - - -	S D $\bar{W}$ L	- - -	- - -	- - -	- - -	- - -	- - -	$\bar{S}$ O W	- - -	-	Search i = DR, write T_7 = 1
A-80	1	- - - -	- - - -	- - -	- - -	- - -	- - -	- - -	S 1 $\bar{W}$	- - -	- - -	-	Search T_6 = 1

A-81 Transfer matched word into DR

	DP	A	B	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	D	
A-82	1	S D $\overline{W}$ L	- - - -	- - -	- - -	- - -	$\overline{S}$ 1 W	- - -	$\overline{S}$ 1 W	- - -	- - -	-	Search j = DR, Write $T_4 = 0, T_6 = 0$

A-83 Jump to A-71

A 84 ⟶ A-116 COST MATRIX ADJUSTMENT

A-84 Set $B_{FL} = d_i$ max, $B_{IL} = d_i$ min

A-85 Set $A_{FL} = d_i$ min, $A_{IL} = d_i$ max

A-86	1	- - - -	- - - -	- - -	- - -	$\overline{S}$ O W	- - -	- - -	- - -	- - -	- - -	-	Clear T3 = 1
A-87	1	S O $\overline{W}$ N	- - - -	- - -	- - -	S 1 $\overline{W}$	S O $\overline{W}$	- - -	- - -	S O $\overline{W}$	- - -	-	Search $D_d = 0$, $T_3 = 1$ $T_4 = 0$, $T_7 = 0$

A-88 IF DP = 0 jump to A-91

A-89 IF DP = 1 jump to A-96

A-90	1	S 1 $\overline{W}$ N	- - - -	- - -	- - -	S 1 W	S O $\overline{W}$	- - -	- - -	S O $\overline{W}$	- - -	-	Search $D_d = 1$, $T_3 = 1$ $T_4 = 0$, $T_7 = 0$
A-91	1	- - - D	- - - -	- - -	- - -	- - -	- - -	- - -	- - -	- - -	- - -	-	

A-92 IF A counter ≠ Initial limit, jump to A-84

A-93	1	- - - -	- - - -	- - -	- - -	S 1 W	- - -	- - -	- - -	- - -	- - -	-	Search $T_3 = 1$ Write $T_3 = 0$

A-94 N Select 1st match

A-95	N	- - - -	- - - -	- - -	- - -	$\overline{S}$ O W	- - -	- - -	- - -	- - -	- - -	-	Write $T_3 = 1$

	DP	A	B	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	D	
A-96	1	- - - -	- - - -	- - -	- - -	- - -	- - -	- - -	- - -	- - -	$\overline{S}$ 1 W	-	Clear $T_8 = 0$
A-97	1	- - - -	S O $\overline{W}$ N	- - -	- - -	S 1 $\overline{W}$	- - -	- - -	- - -	- - -	- - -	-	Search $D_d = 0$ $T_3 = 1$
A-98	IF DP ≠ 0 jump to A-102												
A-99	1	- - - -	S O W N	- - -	- - -	- - -	- - -	- - -	- - -	S 1 $\overline{W}$	S 1 W	-	Search $D_d = 0$ $T_7 = 1$, $T_8 = 1$ Write D_d=1, T_8=1
A-100	1	- - - -	S 1 W I	- - -	- - -	- - -	- - -	- - -	- - -	S 1 $\overline{W}$	S 1 $\overline{W}$	-	Search $D_d = 1$ $T_7 = 8$, $T_8 = 1$ Write $D_d = 0$
A-101	Jump to A-104												
A-102	1	- - - -	S 1 W N	- - -	- - -	- - -	- - -	- - -	- - -	S 1 $\overline{W}$	S O W	-	Search D_d T_7=1, T_8= 0 Write D_d=0, T_8=1
A-103	1	- - - -	S O W I	- - -	- - -	- - -	- - -	- - -	- - -	S 1 $\overline{W}$	S O $\overline{W}$	-	Search $D_d = 1$ T_7= 1, T_8 = 0 Write $D_d = 0$
A-104	IF B counter ≠ Initial limit, jump to A-97												
A-105	1	- - - -	- - - -	- - -	- - -	- - -	- - -	- - -	- - -	- - -	$\overline{S}$ 1 W	-	Clear $T_8 = 0$
A-106	1	- - - -	S O $\overline{W}$ N	- - -	- - -	S 1 $\overline{W}$	- - -	- - -	- - -	- - -	- - -	-	Search $D_d = 0$ $T_3 = 1$
A-107	IF DP ≠ 0 jump to 111												
A-108	1	- - - -	S O W N	- - -	- - -	- - -	- - -	- - -	- - -	S 1 $\overline{W}$	S 1 $\overline{W}$	-	Search $D_d = 0$ T_7 =1, $T_8 = 1$ Write $D_d = 1$

	DP	A	B	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	D	
A-109	1	- - - -	S 1 W 1	- - -	- - -	- - -	- - -	- - -	- - -	S 1 $\overline{W}$	S 1 W	-	Search D_d = 0 T_7=1, T_8 = 1 Write D_d=0, T_8=0
A-110	Jump to A-113												
A-111	1	- - - -	S 1 W N	- - -	- - -	- - -	- - -	- - -	- - -	S 1 $\overline{W}$	S O $\overline{W}$	-	Search D_d = 1 T_7=1, T_8 = 0 Write D_d = 0
A-112	1	- - - -	S - W 1	- - -	- - -	- - -	- - -	- - -	- - -	S 1 $\overline{W}$	S O W	-	Search D_d = 0 T_7=1, T_8 = 0 Write D_d = 1
A-113	IF B Counter ≠ Initial limit, jump to A-106												
A-114	Set B_{FL} = i max, B_{IL} = i min												
A-115	Set A_{FL} = j max, A_{IL} = j min												
A-116	Jump to A-71												
	A-117 → A-131 ASSIGNMENT ADJUSTMENT												
A-117	1	- - - -	- - - -	- - -	- - -	$\overline{S}$ 1 W	- - -	- - -	- - -	- - -	- - -	-	Clear T_3 = 0
A-118	1	- - - -	- - - -	- - -	- - -	- - -	- - -	- - -	- - -	S 1 $\overline{W}$	- - -	-	Search T_6 = 1
A-119	N	Transfer matched word into DR											
A-120	1	S D $\overline{W}$ L	- - - -	- - -	- - -	- - -	$\overline{S}$ O W	- - -	- - -	- - -	- - -	-	Search j = DR Write T_4 = 1
A-121	1	S D $\overline{W}$ L	- - - -	- - -	S 1 $\overline{W}$	$\overline{S}$ O W	- - -	- - -	- - -	- - -	- - -	-	Search j = DR T_2 = 1 Write T_3 = 1
A-122	IF DP = 0, jump to A-127												

	DP	A	B	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	D	
A-123	N	Transfer matched work into DR											
A-124	1	S D $\overline{W}$ L	- - - -	- - -	- - -	- - -	$\overline{S}$ 1 W	- - -	- - -	- - -	- - -	-	Search j = DR Write $T_4 = 0$
A-125	1	- - - -	S D $\overline{W}$ L	- - -	- - -	- - -	- - -	$\overline{S}$ 1 W	- - -	- - -	- - -	-	Search j = DR Write $T_5 = 0$
A-126		Jump to A-119											
A-127	1	- - - -	- - - -	- - -	$\overline{S}$ 1 W	S 1 $\overline{W}$	- - -	- - -	- - -	- - -	- - -	-	Search $T_3 = 1$, Write $T_2 = 0$
A-128	1	- - - -	- - - -	- - -	$\overline{S}$ O W	- - -	- - -	- - -	S 1 $\overline{W}$	- - -	- - -	-	Search $T_6 = 1$ Write $T_2 = 1$
A-129		IF DP = 0, jump to S-1											
A-130	1	- - - -	- - - -	- - -	- - -	$\overline{S}$ 1 W	- - -	$\overline{S}$ 1 W	$\overline{S}$ 1 W	$\overline{S}$ 1 W	- - -	-	Clear $T_3=0, T_5=0$ $T_6=0, T_7=0$
A-131		Jump to A-71											
A-132		Stop											

Tag T_2 contains Assignments

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
General Precision, Inc, Librascope Group.
808 Western Ave, Glendale, Calif

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
N/A

3. REPORT TITLE
Study of Associative Processing Techniques

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Final Report - May 65 - March 66

5. AUTHOR(S) (Last name, first name, initial)
Bird, R.M.
Cass, J.L.
Fuller, Richard H.
Tanner, P.E.
Tu, Dr. J.C.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
September 1966	186	

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
AF30(602)-3756	N/A
b. PROJECT NO. 5581	
c. Task #558109	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d	RADC-TR-66-209, Volume I

10. AVAILABILITY/LIMITATION NOTICES
This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMLI), GAFB, N.Y. 13440

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Rome Air Development Center (EMIID) Griffiss Air Force Base, New York 13440

13. ABSTRACT

This report is in two volumes and describes results of a study of associative processing techniques performed by Librascope Group of General Precision Inc. under RADC contract AF30(602)-3756. Volume I is an unclassified document and contains all the material in the report except that concerning the ELINT reconnaissance problem which is contained in Volume II. Volume II is a classified document at the SECRET level.

DD FORM 1 JAN 64 1473

UNCLASSIFIED
Security Classification

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Associative Processing						
Computers						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year, or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through

_____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through

_____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through

_____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.

Security Classification